テキスト型アプリケーション群を統合する

グラフィカルユーザインタフェースシステム

高　濱　徹　行

1996

正　誤　表

ページ	行	誤	正
p. 118	l. 14	全国大会(5)	全国大会講演論文集(5)

テキスト型アプリケーション群を統合する

グラフィカルユーザインタフェースシステム

高　濱　徹　行

１９９６

概 要

グラフィカルユーザインタフェース(GUI)は，初心者にも使いやすい利用環境を提供することができるが，アプリケーション開発者の立場から見ると，以下のような問題点が存在する。

- GUIはハードウェアやグラフィックスソフトに依存しがちであるため，統一的なインタフェースを提供するのが困難であり，汎用性・移植性が低い。
- GUIアプリケーションの記述は複雑で記述量も多くなるため，開発の期間やコストが増大してしまう。
- 文字入出力を中心とするテキスト型ユーザインタフェース(TUI)を基本とするアプリケーションが有効に利用できない。
- GUIアプリケーションは，文字端末からでは全く利用できない。

本論文では，これらの問題点を解決するために，TUIアプリケーション群を作成し，それらを統合して全体として1つにまとめるGUIプログラムを別に作成するという方法を提案する。すなわち，GUIアプリケーションをTUIアプリケーションとGUIプログラムに分割して作成するというGUI分離型アプリケーションを基本とし，その構築を支援するシステム(以下では統合化支援システムという)を提供するという方法である。このようにTUIを積極的に活用することによって，

- アプリケーションの開発はTUI部までを考慮すればよいため，比較的ハードウェアに依存しない記述が可能となり，汎用性・移植性が高くなる。
- 統合化支援システムにより既存のTUIアプリケーションをGUI環境に適応させることが可能となる。特にUNIXにおいてはGUIに対応していない既存のコマンドが非常に多数存在しているので，このようなコマンドをGUIに適応させることも比較的容易になる。

- GUIを有する端末のユーザはGUI環境でGUI対応のアプリケーションを使用できる一方，文字端末のユーザはTUI対応のアプリケーションをそのまま利用できる。
- 統合化支援システムにおいて，GUI記述用の簡易言語およびGUIビルダを提供することによって開発期間が短縮し，開発コストも軽減される。

本論文では，GUI分離型アプリケーションの構築を支援するための統合化支援システムとして3種類のシステムを提案する。

2章では，宣言型統合化支援システムUAI/Xについて述べる。UAI/Xは，X WindowにGUIを記述するための宣言型簡易言語を提供する。簡易言語では，ウィンドウの階層およびクラスを宣言することによりGUIを定義することができる。各ウィンドウのリソース値やコールバック関数としてTUIアプリケーションの呼び出しを指定する機能を提供しているため，TUIアプリケーションをGUI環境下に統合して1つのGUIアプリケーションを構成することができる。UAI/Xを用いてファイル管理システムを構築することにより，C言語での記述と比較して記述量が約1/8〜1/10に軽減されることも示す。

3章では，手続き型統合化支援システムXTSSについて述べる。XTSSは，X WindowにGUIを記述するための手続き型簡易言語を提供するXTSと，TUIアプリケーション間を結合するための通信路を提供するXTCという2つの部分から構成されるクライアント/サーバ型のシステムである。簡易言語では，X Toolkitに準拠したコマンドによりGUIを定義することができる。さらに，XTCを介してTUIアプリケーション群とXTS間を結合することによりGUI分離型アプリケーションを構築することができる。XTSSを用いてテキストエディタにメニューを付加したシステムを構築することにより，C言語での記述と比較して記述量が約1/2〜1/3に軽減されることも示す。

4章では，視覚型統合化支援システムXTSS Builderについて述べる。XTSS Builderは，CreatorとAnalyzerから構成される。Creatorは，木構造で表示されたウィンドウのクラス階層を参照しながら，視覚的にGUIを定義し，XTS簡易言語を生成するためのシステムであり，簡易言語についての知識を持たないユーザでも簡単に利用することができる。Analyzerは，既存のGUIアプリケーションを参照することにより，XTSあるいはUAI/X簡易言語を自動的に生成するシステムであり，両簡易言語以外で記述されたGUIアプリケーションをXTSSあるいはUAI/Xで再構築するための労力が非常に軽減される。

5章では，CAIシステムをGUI分離型アプリケーションとして作成することを提案する。GUI分離型のCAIシステムの実例として，GUIに対応した知的LISP言語学習システムを構築する。TUIアプリケーションである既存の知的LISP言語学習システムをそのまま用い，

XTSS を利用することにより，ボタン，メニューおよびマルチウィンドウで構成されたGUIアプリケーションとして構築する方法について述べる。

最後に，6 章では，統合化支援システムに残された問題点や将来に向けた計画等について述べる。

謝辞

本研究を進めるに当たり指導や様々な助言をいただきました福井大学の中村正郎教授に感謝致します。

本論文に関して指導や様々な助言をいただきました京都大学の長尾眞教授に感謝致します。

本論文に関して議論し有益な示唆および助言をいただきました京都大学の松山隆司教授並びに上林彌彦教授に感謝致します。

本研究全般について議論し有益な助言をいただきました福井大学の小倉久和教授に感謝致します。

研究を進める過程で色々と励ましていただきました広島修道大学の廣光清次郎教授に感謝致します。

GUI 分離型 CAI システムの構築に協力していただき私的な面でも様々に支援してくれた福井大学の高濱節子助教授に感謝致します。

また，中村研究室の諸氏，特に，研究に関して様々な助言をいただいた柳瀬龍郎講師，松山幸雄技官，UAI/X の初期バージョンの作成に協力していただいた永和システムマネジメントの中谷洋一氏，XTSS Builder の設計・製作に協力していただいた沖電気株式会社の八木正和氏および高嶋技研株式会社の中出雅子氏に感謝致します。さらに，GUI 分離型 CAI システムの構築に協力していただいた酒井化学株式会社の牧野とみ氏に感謝致します。

最後に，長年に渡り温かい目で見守ってくれた両親，高濱末夫・貞子に感謝の意を捧げます。

目 次

表目次

図目次

Chapter 1

はじめに

1.1 グラフィカルユーザインタフェースとその諸問題

現在，ワークステーションおよびパーソナルコンピュータの低価格化・高機能化によりグラフィカルユーザインタフェース (Graphical User Interface, GUI) がかなり普及してきている。X Window や MS-Windows をはじめとする共通的なウィンドウシステムの普及によりこの傾向に益々拍車がかかってきている。確かにアイコン・メニュー・図形等を効果的に用いてビットマップディスプレイ装置とマウスの機能を十分に生かせば，

- 操作対象を直接画面上で指示できるため直感的に分かりやすい
- 対象毎にどの様な操作があるかを覚える必要がない
- イメージを活用するため視覚的に情報を捉えることができる

等の長所があり，初心者にも使いやすいユーザインタフェース環境を提供することができる。[Kuno89]

このようにGUIは従来，計算機の操作方法を視覚的に表現することにより初心者向きの環境を提供してきたが，現在では計算機を高度に利用する分野においても広く使用されるようになってきている。例えば，研究分野では解析データやシミュレーション情報の可視化のために，教育分野ではCAIにおけるユーザフレンドリネスの向上とイメージ情報の活用による学習効果の向上のために，産業分野でもプレゼンテーション等に使用されている。さらに一歩進んで，文字情報・視覚情報・音声情報を総合的に取り扱うマルチメディアユーザインタフェースに関する研究も活発に行なわれている。

確かに，GUI化あるいはマルチメディア化は，ユーザにとっては好ましいが，アプリケーションの開発者の立場から見ると，次のような問題点が存在する。[Takahama90, Takahama91,

Takahama92, Takahama93a, Takahama93b, Takahama94]

1. GUIはディスプレイ装置等の入出力ハードウェアやグラフィックスソフトに依存しがちであるため，統一的なインタフェースを提供するのが困難であり，汎用性および移植性が低い．

2. GUIアプリケーションの記述はかなり複雑である。このため，大量の記述が必要となり，アプリケーション中に占めるユーザインタフェースに関連する部分の割合が大きくなる。また，開発の期間・コストが増大してしまう。[Kuno89, Yokoyama89]

3. 既存のアプリケーション，特に日常的に使用されているコマンド等は，GUI環境に対応しておらず，文字入出力を中心とするテキスト型ユーザインタフェース (Textual User Interface, TUI) を基本としている。したがって，これら既存のアプリケーションをGUI環境に適応させるためには，アプリケーションを再構築しなければならず，既存のアプリケーションが有効に活用できない。

4. GUIの機能を十分に利用したアプリケーションは通常の文字端末からは使用できない。今後次第に文字端末からX Window端末等のGUI環境を提供する端末への移行が進むと考えられるが，現状では文字端末を用いているユーザからもアプリケーションを使用できる状態にしておく必要がある。

1.2 従来の解決方法

前節で述べた問題点に対処するために，従来から様々な方法が提案されている。問題点1. に対処する試みとしては，以下のような方法がある。

- X Windowのように標準のツールキットと規約を定め，アプリケーションの開発者がその規約に準じてツールキットを使用することでインタフェースを作成する方法 [McCormack91, Young89, Aguin92, Kabutogi90, Petzold89]

- ユーザインタフェースを管理する独立のシステム (User Interface Management System, UIMS) を用意する方法 [Yokoyama89, Olsen92]

前者の場合には規約を習得するのが大変であり，ツールキットを使用しても依然としてユーザインタフェース作成の負担が大きいため，問題点2. が解決されない。後者の場合にもアプリケーションとは別にインタフェースの記述を用意し，さらにアプリケーションとUIMS

の整合性を取る必要のあることが指摘されている [Kuno89]. また，いずれの場合でも，既存のアプリケーションをGUI環境に統合するためにはアプリケーション中にGUIのための処理を記述しなければならないため，問題点 3. が解決できない。

問題点 2. を解決する試みには，以下のようなものがある。[Sugimoto89]

- GUI記述用の簡易言語を提供する方法
- CADのような操作でGUIの構築を視覚的に支援するGUIビルダを提供する方法

前者には，OSF/Motifが提供するUIL [OSF90], Common Lisp [Ida91] をベースとするCLX [Scheifler89], オブジェクト指向の機能をLISPに取り入れたXLISP [Almy92] をベースとするwinterp [Mayer90], tcl [Ousterhout90] 言語をベースとするtcl/tk [Ousterhout91], Wafe [Neumann93] など様々な研究がある。後者としては，鼎 [Rekimoto90] やNEXTSTEPのInterfaceBuilderがよく知られている。しかし，いずれにしても主眼はGUI構築に向けられており，TUIを考慮しておらず，問題点 3.,4. が解決される訳ではない。

問題点 3. を解決する試みには，以下のようなものがある。

- 自動的にウィンドウを開いてTUIアプリケーションの出力を表示するコマンドを準備する方法 [Kuno88]
- 既存のTUIアプリケーションを別プロセスで起動しウィンドウとそのプロセス間のインタフェースを直接プログラミングする方法 [Takahashi89, Isoda87]

前者の場合は，基本的にコマンドに対応してウィンドウを開くだけで，他のアプリケーションから呼び出して使用したり，複数のアプリケーションを統合して全体を一つのシステムとして動作させるという点では制約が多い。Xアプリケーションであるxtermを直接用いる場合も同様である。後者の場合には高度な知識と複雑なプログラミングを必要とするというきわめて重大な問題点が残されている。

問題点 4. を解決する方法としては，TUIに対応したアプリケーションを利用する方法と文字端末にも対応したUIMSを構築する方法があるが，後者の方法では既存のアプリケーションをUIMS用に変更する必要があるため，問題点 3. が解決されない。

1.3 グラフィカルユーザインタフェース分離型アプリケーション

本論文では，以上の問題点を解決するために，

- GUI 分離型アプリケーションというGUIアプリケーションの構築形態
- GUI 分離型アプリケーションの構築を支援するための支援システム

を提案する。GUI分離型アプリケーションとは，GUIアプリケーションを，TUIアプリケーション群とそれらを統合して全体として1つにまとめるGUIプログラムから構成しようという考え方である。従来のGUIアプリケーションは，図1.1のようにGUI部を含んで記述される。これに対してここで提案するGUI分離型アプリケーションでは，図1.2のようにTUIアプリケーション群と，GUIプログラムに分離して記述する。

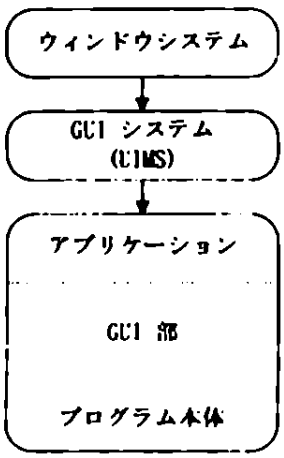

図 1.1: 従来のGUIアプリケーション

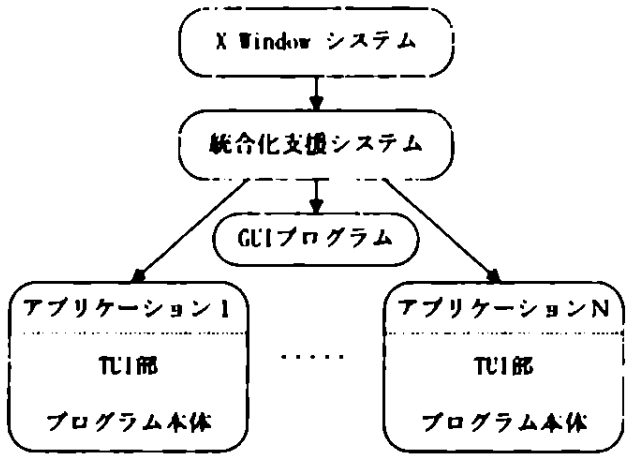

図 1.2: GUI 分離型アプリケーション

GUI分離型アプリケーションを構築する場合の手順は以下のようになる。

1. ユーザインタフェース以外の部分については構築しようとするGUIアプリケーションと同等の機能を持ち，通常の文字端末からでも利用可能なTUIアプリケーションを構築するか，あるいは既存のTUIアプリケーションをそのまま使用する。実現しようとする機能は，1つではなく複数のTUIアプリケーションによって提供してもよい。
2. TUIアプリケーションをGUI環境下に統合するためのGUIプログラムを作成し，全体として一つのGUIアプリケーションとする。なお，TUIアプリケーションのGUI環境への統合を支援するシステムを以下では統合化支援システムと呼ぶことにする。

例えば，GUI対応のプログラミング統合環境を開発する際には，UNIXのCコンパイラであるccコマンドやシンボリックデバッガであるdbxコマンドのようなTUIアプリケーションをまず開発するかあるいは既存のものをそのまま使用し，それらをGUI環境下に統合するGUIプログラムを別途作成するという手順をとることになる。

このようにTUIを積極的に活用することによって，

- アプリケーションの開発はTUI部までを考慮すればよいため，比較的ハードウェアに依存しない記述が可能となり，ユーザインタフェース部の割合もGUI部を直接記述する場合と比較すれば減少するので，問題点1. が軽減される。
- 統合化支援システムにより既存のTUIアプリケーションをGUI環境に適応させることができるので問題点3. が解決する。特にUNIXにおいてはGUIに対応していない既存のコマンドが非常に多数存在しているので，このようなコマンドをGUIに適応させることも比較的容易になる。
- GUIを有する端末のユーザはGUI環境でGUIアプリケーションを使用できる一方，文字端末のユーザはTUI対応のTUIアプリケーションをそのまま利用できるので，TUI部を工夫することにより問題点4. がかなり解決することになる。
- 問題点2. については，統合化支援システムにおいて，GUI記述用の簡易言語およびGUIビルダを提供することによって解決できる。

1.4 統合化支援システム

統合化支援システムでは，以下の点について考慮する必要がある。

GUIの外観の生成 GUIの外観を生成するためにどのような言語あるいは操作を提供するか。

ユーザとの対話的処理 ユーザがGUI上で操作を行なった際の各種対話的処理をどのようにして記述するか。特に，TUIアプリケーションを統合するためにどのような機構を提供するか。

1.4.1 GUI構築支援システム

GUIの外観の生成とユーザとの対話的処理は，一般的なGUI構築支援システムでも考慮する点である。この2つの点に基づき，GUI構築支援システムを分類すると共に，それぞれの特徴について述べる。

- 手続き型システム
  GUIの外観の生成や対話的処理のために必要な手続きや命令を記述することにより，

GUI アプリケーションを構築するシステムである。一般に記述は容易ではないが，特に対話的処理に対する記述の自由度は大きい。このようなシステムとしては前述した X Toolkit, UIL, CLX, winterp, tcl/tk, Wafe などがある。

- 宣言型システム
  GUI の外観を生成するための命令を記述するというよりはむしろ，キーワードやタグにより外観の状態を記述するシステムである。このようなシステムとしては，利用目的は異なるが，WWW(World Wide Web) における HTML(Hyper Text Markup Language) [Lemay95] がある。外観の記述は比較的容易であるが，対話的処理に関する部分を宣言的に記述することは容易ではないため，HTML では CGI(Common Gateway Interface) を利用して外部プログラムを使用したり，java 言語 [Hoff96] を取り込むことによって対話的処理を実現している。

- 視覚型システム
  GUI の外観の生成を視覚的に支援するシステムである。このようなシステムとしては前述した鼎, InterfaceBuilder などがある。視覚的に指定するため，外観の記述は非常に容易であるが，対話的処理に関する部分については視覚的に指定することは困難であり，通常は手続き型のシステムで記述することになる。

1.4.2 テキスト型アプリケーションの統合

ユーザとの対話的処理のうち，TUI アプリケーションの統合の部分について考察するために，対象となる TUI アプリケーションを 2 つの視点から分類する。第一の視点は，アプリケーションの起動形式であり，以下のような 3 種類に大別できる。

- 単純型：必要とされる度に起動される。
- 繰り返し型：時間の変化等の何らかのイベントに応じて繰り返し起動される。
- 連続型：一度だけ起動され，連続的（対話的）に処理を進める。

第二の視点は，ウィンドウから見たアプリケーションの動作であり，以下のような 4 種類に分類できる。

- 作用動作：出力を必要とせずシステムに何らかの副作用を起こす。
- 出力動作：アプリケーションからの出力をそのまま，またはメーターやグラフとしてウィンドウに表示する。

- 入力動作：ウィンドウからの文字入力あるいはメニュー選択等の内容をアプリケーションに対するパラメータあるいは入力データとして用いる。

- 複合動作：入力動作と作用動作あるいは出力動作を組み合わせた動作を行なう。

UNIX コマンドを例にとれば，ホームディレクトリに移動するために cd コマンドを実行するのが単純型作用動作，ファイル名を表示するために ls コマンドを呼び出すのが単純型出力動作，date コマンドを定期的に呼び出しディジタル時計を表示するのが繰り返し型出力動作である。ダイアログウィンドウでファイル名を指定し rm コマンドでそのファイルを削除するのが単純型複合（入力－作用）動作，対話型の簡易計算プログラムである bc コマンドを応用して電卓を構成するのが連続型複合（入力－出力）動作である。通常入力動作は，作用または出力動作と組み合わせることにより複合動作として用いられる。

統合化支援システムでは，この型および動作という 2 つの視点の組合せで分類されるアプリケーションの各範疇に応じた統合機構を提供する必要がある。

1.4.3 提案する統合化支援システム

本論文では GUI 分離型アプリケーションの構築を支援する統合化支援システムを提案するが，1.4.1 節で述べた 3 タイプのシステムはそれぞれに異なる特長を有しているため，各タイプのシステムを個別に構築する。

宣言型システムとしては，あらかじめ定められたキーワードに対する値を記述することにより GUI の外観の生成および対話的処理，ここでは TUI アプリケーションとのやりとり，を記述する UAI/X を提案する。

手続き型システムとしては，X Toolkit に準拠した命令を提供する XTS と，TUI アプリケーションを統合する XTC という 2 つのシステムから構成される XTSS を提案する。UAI/X では GUI の外観を生成する機能と対話的処理を 1 つのシステムで実現しているが，XTSS では GUI の外観を生成する機能を XTS で，TUI アプリケーションの統合に必要な機能を XTC で，というように機能を分割して実現している。

視覚型システムとしては，GUI の外観を構成する部品の親子関係を木構造上で視覚的に指定する XTSS Builder を提案する。XTSS Builder は，GUI プログラムを視覚的に編集するための Creator と，既存の GUI アプリケーションから GUI プログラムを抽出する Analyzer から構成される。Analyzer は UAI/X と XTSS の両方に対応するが，Creator は XTSS のみに対応している。Creator および Analyzer は，GUI の外観の指定に必要な記述を生成でき

るが，対話的処理の部分の記述は生成できない。したがって，対話的処理の部分は直接 GUI プログラムを編集する必要がある。

上記システムの概要とその相互関係を図 1.3 に示す。なお，これらのシステムは 1.4.2 節で考察した各タイプに応じた TUI アプリケーションの統合機構を提供しているが，これについては各システムを説明する際に述べる。

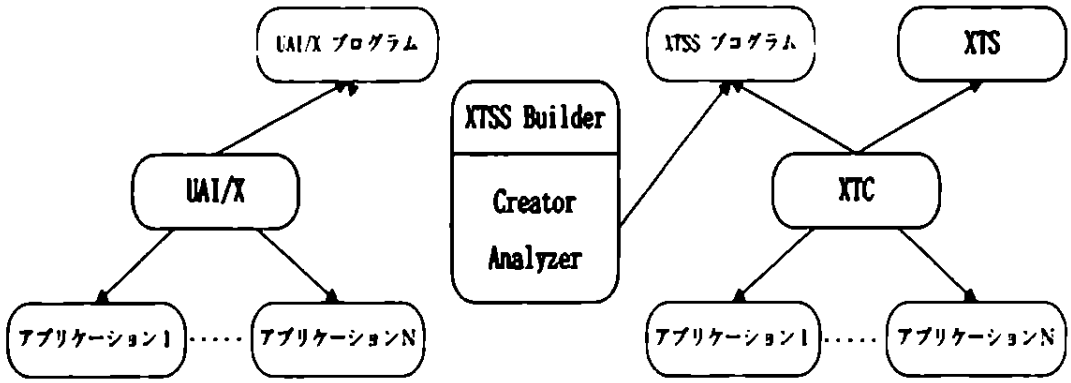

図 1.3: システム概要および相互関係

1.5 論文の構成

本論文では，GUI 分離型アプリケーションの構築を支援するための統合化支援システムとして 3 種類のシステムを提案する。

2 章では，宣言型統合化支援システム UAI/X [Takahama90, Takahama91, Takahama92] について述べる。UAI/X は，X Window に対応した GUI を記述するための宣言型簡易言語を提供する。簡易言語では，ウィンドウの階層およびクラスを宣言することにより GUI を定義することができる。各ウィンドウのリソース値やコールバック関数として TUI アプリケーションの呼び出しを指定する機能を提供しているため，TUI アプリケーションを GUI 環境下に統合して 1 つの GUI アプリケーションを構成することができる。UAI/X を用いてファイル管理システムを構築することにより，C 言語での記述と比較して記述量が約 1/8〜1/10 に軽減されることも示す。

3 章では，手続き型統合化支援システム XTSS [Takahama93a, Takahama93b, Takahama94] について述べる。XTSS は，X Window に対応した GUI を記述するための手続き型簡易言語を提供する XTS と，TUI アプリケーション間を結合するための通信路を提供する XTC という 2 つの部分から構成されるクライアント/サーバ型のシステムである。簡易言語では，X Toolkit に準拠したコマンドにより GUI を定義することができる。さらに，XTC を介し

てTUIアプリケーション群とXTS間を結合することによりGUI分離型アプリケーションを構築することができる。XTSSを用いてテキストエディタにメニューを付加したシステムを構築することにより，C言語での記述と比較して記述量が約1/2〜1/3に軽減されることも示す。

4章では，視覚型統合化支援システムXTSS Builder [Takahama93c, Nakade95]について述べる。XTSS Builderは，CreatorとAnalyzerから構成される。Creatorは，木構造で表示されたウィンドウのクラス階層を参照しながら，視覚的にGUIを定義し，XTS簡易言語を生成するためのシステムであり，簡易言語についての知識を持たないユーザでも簡単に利用することができる。Analyzerは，既存のGUIアプリケーションを参照することにより，XTSあるいはUAI/X簡易言語を自動的に生成するシステムであり，両簡易言語以外で記述されたGUIアプリケーションをXTSSあるいはUAI/Xで再構築するための労力が非常に軽減される。

5章では，CAIシステムをGUI分離型アプリケーションとして作成することを提案する[Sakai94, Takahama97]。GUI分離型のCAIシステムの実例として，GUIに対応した知的LISP言語学習システムを構築する。TUIアプリケーションである既存の知的LISP言語学習システムをそのまま用い，XTSSを利用することにより，ボタン，メニューおよびマルチウィンドウで構成されたGUIアプリケーションとして構築する方法について述べる。

最後に，6章では，統合化支援システムに残された問題点や将来に向けた計画等について述べる。

Chapter 2

宣言型統合化支援システム: UAI/X

UAI/X は，GUI を定義するための宣言型簡易言語を提供する。簡易言語では，ウィンドウの親子関係およびウィンドウの種類・形状・表示位置などのリソース (属性) を宣言的に記述することにより GUI の基本的な構成を指定する。また，“ウィンドウのリソースおよびリソース値を変数および変数値として扱う” という視点を導入することにより，リソースを表現する変数に代入・参照操作を行なえば，ウィンドウのリソース値の設定・変更や参照を行なうことが可能となる。さらに，リソースの記述中にアプリケーションの呼び出しを埋め込む機能を提供することにより，TUI アプリケーション群を統合する際に各アプリケーション用のウィンドウの生成やウィンドウに対する入出力処理を直接プログラミングする必要がなくなり，TUI アプリケーション群を GUI 環境下に統合するための労力が大幅に軽減できることになる。

2.1 宣言型統合化支援システムの構想

統合化支援システムにおいては，GUI の外観の生成とユーザとの対話的処理という２つの機能を提供する必要がある。宣言型統合化支援システムにおいては，これらの機能をキーワードやタグを利用して宣言的に記述することになる。

まず，外観の生成について考察する。外観は，GUI の構成部品である各ウィンドウに対して，以下の２つの要素を指定することにより決定される。

- リソース

  ウィンドウの種類 (クラス)・形状・表示位置あるいは表示する文字列などのようなウィンドウの持つ様々な属性を指定する。

- 親子関係

ウィンドウのクラスによって親となれるかどうか，そして子がどのように配置されるかが決まる。このようなウィンドウ相互間の包含関係を指定する。

これらの要素を宣言的に記述する方法として，記述の一様性に配慮して，キーワードに対して値を対応付けるという形式を採用する。キーワードに対して1つの値が対応する場合には，

キーワード: 値

という形式で，1つ以上の値が対応する場合には，

```
キーワード:
    値1
    ...
    値n
end
```

という形式で記述する。このようにすれば，リソースは，

リソース名: リソース値

で記述でき，親子関係は，キーワード window および winend を用いて，

```
window: 親ウィンドウの名前
    window: 子ウィンドウの名前
    winend
winend
```

で記述できるようになる。なお，end および winend は値のない特殊なキーワードであると考えることができる。

次に対話的処理について考察する。対話的処理は，TUIアプリケーションとのやりとりを行なうための処理である。TUIアプリケーションは1.4.2節で述べたように分類されるため，各範疇に応じた記述方法を提供する必要がある。

- 単純型作用動作の場合

  callback: TUIアプリケーションの呼び出し

と記述する。

- 単純型出力動作の場合
  TUIアプリケーションの出力を表示したいウィンドウの対応するリソースに，

  リソース名: !TUIアプリケーションの呼び出し

  と記述する。

- 単純型入力動作の場合
  ユーザが入力したデータ等は入力を行なったウィンドウのあるリソースの値となっている。この値の参照を可能とするために，"ウィンドウのリソースおよびリソース値を変数および変数値として扱う"という視点を導入する。すなわち，

  @ウィンドウ名 [リソース名]

  によって，そのウィンドウのリソース値を参照するのである。このような変数の記述を他のリソースの値やTUIプログラムのパラメータとして指定することにより入力動作を実現する。

- 単純型複合動作の場合
  上記の動作を組み合わせることにより実現する。

- 繰り返し型の場合
  上記の各動作を繰り返して行なうために，repeat キーワードを導入する。

  repeat: 繰り返しのための条件の指定

  により，repeat キーワードを記述したウィンドウの動作を条件にしたがって繰り返し行なうように指定する。

- 連続型の場合
  連続型では，TUIアプリケーションから出力を受け取るだけではなく，TUIアプリケーションへデータを渡す必要がある。そこで，このようなアプリケーションのために特に application キーワードを導入する。

  application: アプリケーション名
  　TUIアプリケーションの呼び出し

と記述しておき，アプリケーションからの出力は，

●アプリケーション名［入出力モード］

で参照する。アプリケーションへのデータの受け渡しは，

●アプリケーション名［入出力モード］■データ

で実現する。

2.2 システムの概要

UAI/X の全体構成を図 2.1 に示す。影付きの部分はユーザが定義する部分である。

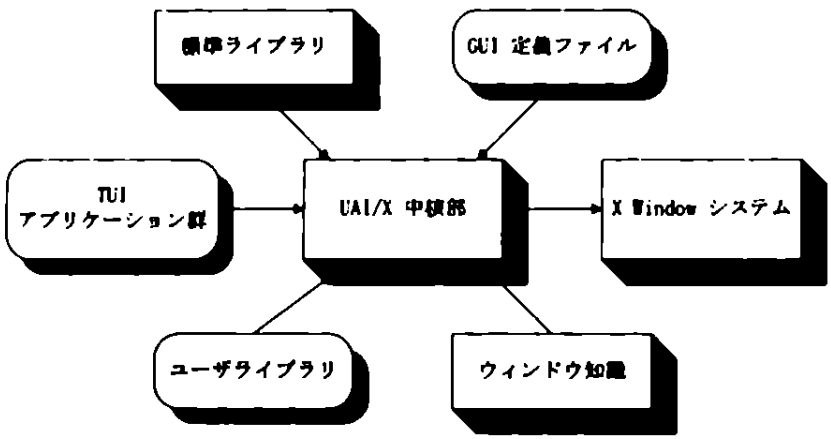

図 2.1: UAI/X の全体構成

UAI/X は，以下の 7 つの部分により GUI を実現する。

1. UAI/X 簡易言語により GUI を記述する GUI 定義ファイル。GUI プログラムに相当する。
2. GUI 定義ファイルを解析し GUI を実現する UAI/X 中核部。
3. TUI アプリケーションを統合する際に頻繁に使用される機能を提供するため準備されている標準ライブラリ。
4. アプリケーション特有の機能を実現するためにユーザが準備するユーザライブラリ。

5. 統合されるTUIアプリケーション群。

6. ウィンドウのクラスの一覧，各ウィンドウの上位クラス，各クラスに属するリソース名，リソースの型，文字列を実際のリソース値に変換するための変換関数という情報を保持するウィンドウ知識。

7. X Window システム。Athena Widget [Peterson91] の機能を利用している。

GUI 定義ファイルにはUAI/X 簡易言語を用いて GUI を記述するが，言語の記述は，以下のような５つの部分に分かれている。

1. 変数定義部：内部変数を定義し必要に応じて初期値を与える。

2. ウィンドウ定義部：ウィンドウの機能・リソース・親子関係，アプリケーションの呼び出しを定義する。

3. トランスレーション定義部：マウスやキー入力などのイベントによって何らかの動作(アクション)を行なう際のイベントーアクションの対応表を定義する。

4. アクション定義部：実際に行なうアクションとして呼び出すアプリケーションを定義する。

5. アプリケーション指定部：連続型のアプリケーションに対してそのアプリケーションとの通信領域を定義する。

なお，簡易言語の詳細については次節で述べる。

中核部は，GUI 定義ファイルに記述された内容に従って，各種イベントに対処するための準備を行ない，実際にウィンドウを開く。以下に処理の流れの概略を記述する。(図 2.2 参照)

1. GUI 定義ファイルの記述にファイルの挿入やマクロ定義を行なう C 言語のプリプロセッサの文法を付与するため，C プリプロセッサを起動する。

2. GUI 定義ファイルを解析し，

   (a) 定義された内部変数を初期化する。

   (b) ウィンドウのクラス，リソース指定，階層関係，およびトランスレーション定義，アクション定義，アプリケーション定義をウィンドウ情報として記憶するとともに必要に応じて変数を自動生成する。

3. ウィンドウ情報に基づきウィンドウの階層を辿り，

   (a) 各ウィンドウについて対応するクラスに関するウィンドウ知識を参照し，リソース値を指定する文字列を実際のリソース値に変換し，ウィンドウを生成する。そのクラスで未定義のリソースについては上位クラスを参照し，上位クラスでも未定義のリソースが存在する場合には未定義エラーとし，そのリソースの定義を無視する。

   (b) トランスレーション定義は X Window の機能を利用し，イベント駆動型の処理として登録する。

4. アプリケーション指定部で定義されたアプリケーションをバックグラウンドプロセスとして起動する。

5. 解析結果に基づきウィンドウを表示する。

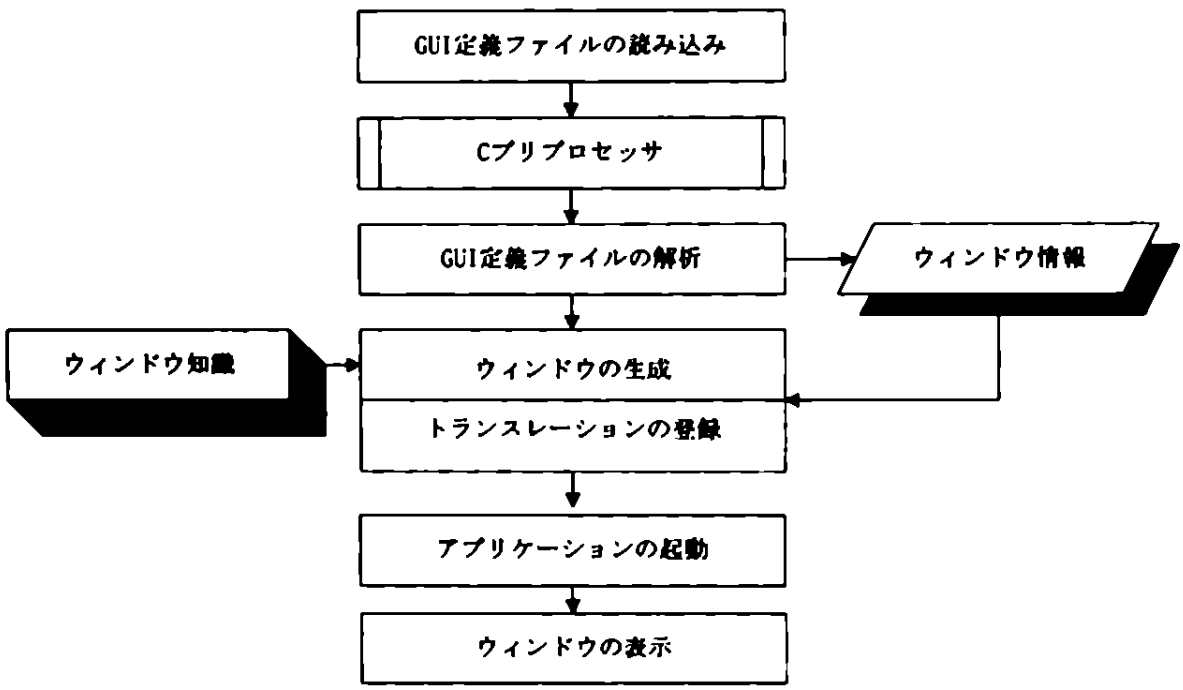

図 2.2: 中核部の処理の流れ

2.3 UAI/X の言語表現

ウィンドウを利用する際には，その利用目的に応じた様々な機能を記述する必要があるだけでなく，形状・表示位置のようなウィンドウのリソースについても記述する必要がある。UAI/X 簡易言語はこれらを記述するための言語であり，変数定義部，ウィンドウ定義部，トランスレーション定義部，アクション定義部，アプリケーション指定部の５つの部分からなる。以下では各部について説明する。

2.3.1 変数定義部

UAI/X では変数を用いてウィンドウとアプリケーション間の通信を行なうが，この変数はウィンドウやアプリケーションを定義することにより自動的に生成される。その他に変数が必要な場合，変数定義部に変数名と必要ならばその初期値を定義する。表記方法は，

```
variable:
    変数名=初期値
    変数名=
```

である。変数値を参照する時には，@変数名 と記述する。

2.3.2 ウィンドウ定義部

ウィンドウにはメニューを表示するメニューウィンドウ，ユーザからの入力を受け付けるダイアログウィンドウ，他のウィンドウから呼び出されるポップアップウィンドウのように様々の種類があり，表示位置，形状，フォントなどのリソースを持つ。ウィンドウのクラスはクラス指定によって選択し，リソースはリソース名に対するリソース値を記述することによって指定する。表記方法を以下に示す。

```
window: ウィンドウ名
    class: ウィンドウのクラス（種類）
    リソース名 1: リソース値 1
      .....
    リソース名 n: リソース値 n
winend
```

クラスおよびリソースとしては，Athena Widget で定義されたものが利用できる。クラスの例としては，メニューを定義する menu クラス，ボタンを定義する command クラス，ダ

イアログウィンドウを定義するdialogクラスなどがあり，クラス指定が省略された場合には親ウィンドウのクラスが継承される。リソース名の例としては，ウィンドウの大きさを指定するwidth, height，表示位置を指定するx, y，フォント名を指定するfontなどがあり，省略時にはX Windowの暗黙値が使用される。ウィンドウはさらに子ウィンドウを持つことがあるが，window:～winendまでを入れ子にすることによって記述する。実際のウィンドウの記述例およびその表示結果を示す。(図 2.3 参照)

```
window: Dialog
    class: dialog
    label: Input?
    value:
    window: Ok
        class: command
    winend
    window: Cancel
        class: command
    winend
winend
```

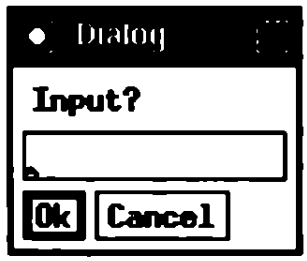

図 2.3: ダイアログウィンドウの表示例

ウィンドウ名を指定した場合には，そのウィンドウ名を変数名とする変数 (ウィンドウ変数と呼ぶ) が自動的に生成され，そのウィンドウのリソース値を @ウィンドウ名 [リソース名] で設定・参照することができる。

2.3.3 トランスレーション定義部

マウスのウィンドウへの出入りによってウィンドウの状態を変化させたり，マウスのクリックによって新たなウィンドウを開きたい場合がある。この様なイベントとアクションとの関係は，以下のように記述する。

translation: トランスレーション名
　イベント系列: アクション列
　　.....
　イベント系列: アクション列

この定義はX Windowにおけるトランスレーションの指定と同じであり，X Windowのトランスレーション機能を利用している。イベント系列の例としては，マウスがウィンドウ内へ移動したことを表わすEnter，右ボタンの押下を表わすBtn1Downなどがある。なお，どのウィンドウでどのトランスレーションを採用するかは，ウィンドウ定義部においてtranslationリソースのリソース値を指定し，

translation: トランスレーション名

で記述する。

2.3.4 アクション定義部

トランスレーション定義部においてイベント系列に対応するアクション列を指定するが，本システムでは，以下のようなアクションが定義されており，動的なウィンドウの表示・非表示および状態変更を容易にしている。

- Refresh(ウィンドウ名): アプリケーションの再起動を行なうことによってウィンドウの状態を最新のものにする。
- Popup(ウィンドウ名): ウィンドウをポップアップする。
- Popdown(ウィンドウ名): ウィンドウをポップダウンする。
- Quit(): システム全体を終了させる。

X Windowで定義されていないユーザ定義のアクションを追加するには，通常関数を定義しアクション登録を行なわねばならないが，本システムではアプリケーションの呼び出しの列をアクションとして呼び出す仕組みを提供している。アクションの定義は，以下のように記述する。

action: アクション名
　アプリケーションの呼び出し
　　.....
　アプリケーションの呼び出し

トランスレーション定義部で,

イベント系列: Action(アクション名)

と記述することでアプリケーション群で記述された新しいアクションを呼び出すことができる。

2.3.5 アプリケーション指定部

連続型のアプリケーションに対しては，アプリケーションをバックグラウンドプロセスとして起動し，そのプロセスと通信を行なうことによって処理を進めて行く必要がある。このようなアプリケーションの定義は，以下のように記述する。

application: アプリケーション名
連続型アプリケーションの呼び出し

これにより自動的にアプリケーション名を変数名とする変数が生成される。変数を参照すれば，対応するアプリケーションから読み込んだ文字列が変数の値となる。変数へ文字列を代入すれば，その文字列をアプリケーションに書き込むことができる。アプリケーションとの入出力の同期については，一行単位の入出力を行なう比較的単純なアプリケーションを統合することを目標とし，一定時間内にアプリケーション側から返されてきた出力を一まとまりのデータとして取り扱うという単純な方法を採用した。@アプリケーション名[line] で一行単位の参照・代入が可能である。

2.3.6 ライブラリ

頻繁に使用される手続きは標準ライブラリとして提供しており，非連続型のアプリケーションと同様に利用することができる。標準ライブラリとしては，アクションに対応して，以下のものがある。

- refresh ウィンドウ名。
- popup ウィンドウ名。
- popdown ウィンドウ名。
- quit。
- toggle 変数名 値: 呼び出されるたびに現在の変数値と値を入れ換える。

ライブラリとしてはユーザが準備するユーザライブラリも存在する。これは必ずしも関数としてではなく，実行可能モジュールとして準備してもよい。

2.4 アプリケーションの統合

以上5つの部を利用することにより，各タイプのTUIアプリケーションがどの様にして統合されるかを具体例を挙げて述べる。

単純型 ウィンドウが活性化されたときにアプリケーションを呼び出す単純型作用動作の場合には，

```
callback: 非連続型アプリケーションの呼び出し
```

と記述する。アプリケーションの出力をリソース値として使用する単純型出力動作の場合には，

```
リソース名: !非連続型アプリケーションの呼び出し
```

と記述することにより，アプリケーションが実行されその出力がリソース値として扱われる。lsコマンドの出力（ファイル名のリスト）を表示する例を図2.4に示す。

```
window: List
    class: list
    list: !ls
winend
```

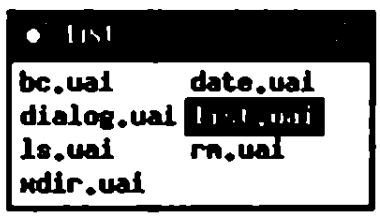

図 2.4: 単純型出力動作の例

ウィンドウから何らかの入力を受け，その入力を用いてアプリケーションを起動する場合，すなわち，単純型複合動作の場合には，単純型作用動作および単純型出力動作においてアプリケーションに対するパラメータとしてウィンドウ変数を用いる。ダイアログウィンドウからファイル名を得，そのファイルを削除する例を図2.5に示す。

```
window: Rm
    class: dialog
    label: File name?
    value:
    window: Ok
        class: command
        callback: rm @Rm[value]
    winend
winend
```

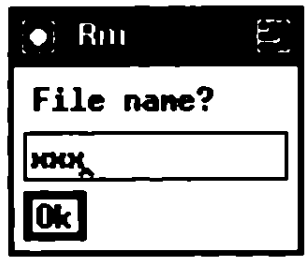

図 2.5: 単純型複合（入力ー作用）動作の例

繰り返し型 繰り返しのためのイベントとしては，現在のところ時間経過のみを想定している。n 秒毎に単純型の動作を繰り返したい場合には，

```
repeat: time n
```

のように記述する。date コマンドを用いてディジタルクロックを構成する例を図 2.6 に示す。

```
window: Date
    class: label
    label: !date
    repeat: time 1
winend
```

図 2.6: 繰り返し型出力動作の例

連続型 アプリケーション指定部でアプリケーションを指定し，アプリケーション変数に対する代入・参照によってウィンドウとの連携を実現する。以下に計算プログラムbcを用いてダイアログウィンドウから計算式を入力し，ボタンがクリックされた時に結果を表示する例を図 2.7 に示す。

```
application: bc
    bc

window: Bc
    class: paned
    window: Input
        class: dialog
        label: Enter Expression?
        value:
    winend
    window: Output
        class: label
        label: @bc[line]
    winend
    window: Do calc
        class: command
        callback:
            bc[line]=@Input[value]
            refresh Output
        end
    winend
winend
```

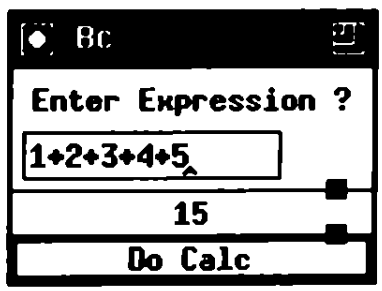

図 2.7: 連続型複合（入力－作用）動作の例

その他 補助的な動作として、直接アプリケーションを呼び出すのではなく、アプリケーションの呼び出しの際に必要な情報を保持する情報保持動作が存在する。この場合にはウィンドウの定義のみを行ない、内部変数を利用すればよい。ls コマンド呼び出しの際のオプションを変更する例を図 2.8 に示す

```
variables:
    option=
    label=NoDot

window: Ls
    class: paned
    window: Label
        class: command
        label: @label
        callback:
            toggle option -a
            toggle label Dot
            refresh Label Files
        end
    winend
    window: Files
        class: list
        list: !ls @option
    winend
winend
```

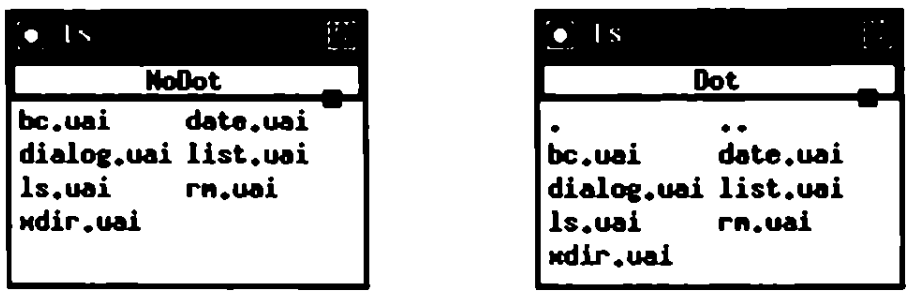

図 2.8: 情報保持動作の例

また、リソース値などによりシステムの動作を変更させる場合には、条件分岐として

条件 ? 条件成立時の動作 : 条件不成立時の動作

ウィジェット変数 [リソース名] ?= 値

という記述を用いることもできる。前者は条件により動作を選択する単純な条件分岐である。後者は，値が空でなければリソースに値を代入し，ウィジェット変数に対応するウィンドウをポップアップすることを指定する。

2.5 UAI/X の応用

UAI/X の応用例として，UNIX のファイル操作関連のコマンドである ls, cp, mv, rm などを統合し，X11R4 [Peterson89, McCormack88] のクライアントとして提供されている xdir と同等のファイル管理システム（xdir/UAI/X と呼ぶ）を構築した例を示す。

xdir の最上位は 図 2.9 のようにメニュー部・ディレクトリ名表示部・ファイル一覧表示部からなる。

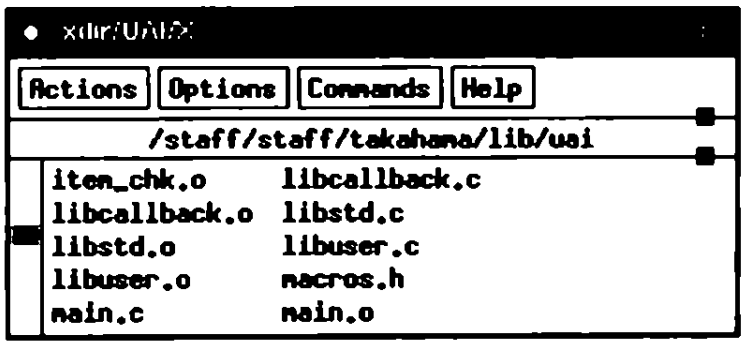

図 2.9: xdir 最上位の表示

UAI/X では以下のように記述できる。なお，以下のリストはその概略である。

```
window: xdir/UAI/X
    class: paned
    window: /* メニュー部 */
        class: box
        window: Actions
            class: menuButton
            .....
    winend
    window: working_directory /* ディレクトリ名表示部 */
        class: label
```

```
        label: !pwd
    winend
    window: /* ファイル一覧表示部 */
        class: viewport
        window: Files
            class: list
            list: !ls @lsoption @rest
            translation: ShortCut
        winend
    winend
winend
```

ディレクトリ名，ファイル名を表示するためにそれぞれpwd, lsというTUIアプリケーションを呼び出している。マウスのクリックでファイル名の指定が可能であるため，コマンドラインから直接操作するのに比べて操作性が大きく向上している。

サブメニューの例としてCommandsサブメニュー中のRename(名前の付け替え)について説明する。(図2.10参照)

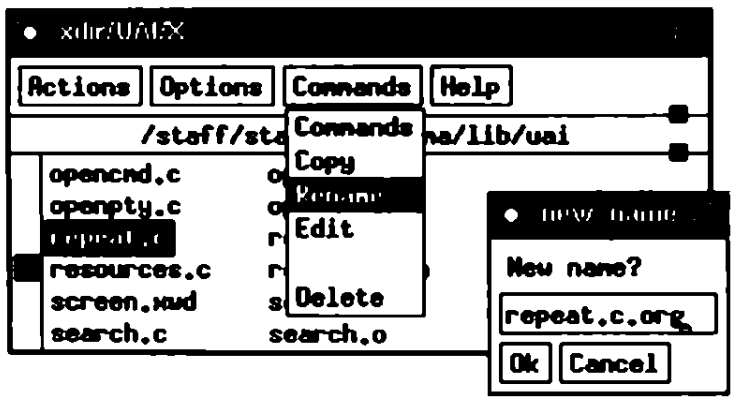

図 2.10: サブメニュー部の表示

この項目が選択されればnew_nameウィンドウが表示され，新しいファイル名を入力し，Okボタンを押すとmvコマンドが実行され，ファイル一覧表示部が更新される。Cancelボタンを押した場合には実行を中止する。また，mv実行中にエラーが起きた場合には，errorウィンドウ中にエラーメッセージが表示される。なお，エラーメッセージを表示する際に条件分岐を用いている。

2.6 評価

キーワードとそれに対する値を宣言的に記述することでGUIの外観の指定が可能となることを示した。また，対話的処理についてもキーワード callback, repeat, application および ! の導入，ウィンドウのリソースや連続型TUIアプリケーションに対する入出力を変数として扱う視点の導入により宣言的に記述できることを示した。

UAI/X の特徴として，以下のような点が挙げられる。

- TUIアプリケーションの統合

  一般に，プログラミング言語からTUIアプリケーションを呼び出すには，プロセス間通信など特殊な知識が必要となり，GUIを定義するプログラマには大きな負担となる。これに対してUAI/X では，"ウィンドウの属性および属性値を変数として扱う"という視点に基づき，GUIにおけるウィンドウの親子関係やリソース等を記述し，リソースの定義中にTUIアプリケーションの呼び出しを埋め込むことにより，TUIアプリケーション群を1つのGUIアプリケーションとして統合する機能を提供している。

  この機能の有用性は，UAI/X を用いてC言語で記述されたファイル管理システムxdirと同等の機能を持つシステムを容易に構築できることにより示した。

- GUI定義に要する知識の少なさ

  X Window においてGUIを定義するためには，最低限，GUIを構成する部品であるウィンドウに関する知識およびウィンドウにおける各種リソースに関する知識が必要である。通常これらの知識に基づいてGUIを記述するが，このためには何らかのプログラミング言語を使用する必要がある。GUIを記述するために最も良く使用されているC言語を例にとると，GUIを定義するには，

  - ウィンドウ(ウィジェット)用変数の宣言
  - ウィジェット生成用関数の呼び出し
  - リソース設定・参照用関数の呼び出し

  などを記述しなければならない。これに対して，UAI/X では

  - ウィンドウ用変数は自動的に生成されるため，変数の宣言は不要である。
  - ウィンドウは宣言的に記述するため，関数を呼び出す必要がない。

- リソースの設定・参照は，自動的に生成されるウィンドウ変数を介して行なうため，関数の呼び出しは不要である。

という利点がある。なお，UAI/X 以外の簡易言語でも，多くの場合ウィジェット用変数の宣言は不要となっている。

- GUI 定義に要する記述量の少なさ
  UAI/X では，GUI の定義や TUI アプリケーションの呼び出しを宣言的に記述できるため，記述量がかなり少なくなる。前節で取り上げた GUI アプリケーションを C 言語で直接記述した場合に対する記述量の比較を表 2.1 に示す。UAI/X を用いれば約 1/8 から 1/10 の記述量でシステムが構築できることが分かる。(図 2.11〜2.15 参照)

表 2.1: xdir と xdir/UAI/X の記述量の比較

システム	xdir	xdir/UAI/X
言語	C 言語	UAI/X 簡易言語
行数	1,652	201
バイト数	36,843	3,700

また，xdir/UAI/X をより使用し易くするためにシェルの呼び出しを追加してみたが，そのために必要な修正は 4 行の追加だけであった。このことも UAI/X の記述力の高さを示している。

- 対話的な GUI 記述環境
  UAI/X はインタプリタ系の言語であり，命令が解析あるいは実行される際に文法エラーや実行時エラーの表示を行なうため，C 言語などのコンパイラ系の言語と比較してデバックが非常に容易である。なお，LISP 等のインタプリタをベースとする簡易言語も同様の特徴を持っている。

UAI/X は比較的単純なアプリケーションを簡単な記述で統合することを目指したため，

- 新しいウィジェットの定義
- 新しいリソースの定義
- クライアント間の通信
- グラフィックスプリミティブの直接使用

- 動的なウィジェットの生成・削除

の機能は提供していない。これらは主として新しいウィジェットを定義するいわゆるウィジェットプログラマの範囲であり，C 言語で記述する必要がある。これまでの議論から明らかなように，ウィジェットとして定義されれば UAI/X に組み込むのは非常に簡単である。

```
variable:
        lsoption=-1F    /* Options メニューの dotfiles コマンドのために必要 */
        rest=           /* Options メニューの backupfiles コマンドのために必要 */
        view=view
        edit=sj2 -e vi
        about=/usr/local/lib/uai/xdir.about
        help=/usr/local/lib/uai/xdir.help
        dotfiles=DotFiles
        backupfiles=BackupFiles
end

translation: ShortCut
        <Btn1Up>(2): Set() Notify() Activate(ChangeDir)
        <Btn3Up>: Activate(ParentDir)
        <Btn2Up>(2): Activate(Quit)
end

window: xdir/UAI/X
  class: paned
  width: 330
  window:
    class: box
    window: Actions
      class: menuButton
      window:
        class: simpleMenu
        label: Actions
        window:
          class: smeBSB
          label: View File
          callback: jterm -e @view @list[value] &
        winend
        window: ChangeDir
          class: smeBSB
          label: Change Directory
          callback:
            cd @list[value]
            refresh working_directory list
          end
        winend
        window: ParentDir
          class: smeBSB
          label: Parent Directory
          callback:
            cd ..
            refresh working_directory list
          end
        winend
```

図 2.11: xdir/UAI/X の記述 (その 1)

```
    window:
      class: smeBSB
      label: About Xdir
      callback: xterm -e @view @about &
    winend
    window:
      class: smeBSB
      label: Update List
      callback: refresh list
    winend
    window: Quit
      class: smeBSB
      label: Quit
      callback: quit
    winend
  winend
winend
window: Options
  class: menuButton
  window:
    class: simpleMenu
    label: Options
    window:
      class: smeBSB
      label: @dotfiles
      callback:
        toggle lsoption -aF
        toggle dotfiles NoDotFiles
        refresh list
      end
    winend
    window:
      class: smeBSB
      label: @backupfiles
      callback:
        toggle rest | grep -v ~$
        toggle backupfiles NoBackupFiles
        refresh list
      end
    winend
  winend
winend
```

図 2.12: xdir/UAI/X の記述 (その 2)

```
    window: Commands
      class: menuButton
      window:
        class: simpleMenu
        label: Commands
        window:
          class: smeBSB
          label: Copy
          callback: popup where
          window: where
            class: dialog
            x: 0
            y: 0
            label: Where?
            value:
            grab: nonexclusive
            window:
              class: command
              label: Ok
              callback:
                error[value]?=!cp @list[value] @where[value]
                refresh list
                popdown
              end
/* mv,cp,rmdir がエラーメッセージを出力すると表示される */
              window: error
                class: dialog
                x: 0
                y: 0
                label: Error
                value:
                grab: exclusive
                window:
                  class: command
                  label: Oh, well
                  callback: popdown
                winend
              winend
            winend
            window:
              class: command
              label: Cancel
              callback: popdown
            winend
          winend
        winend
```

図 2.13: xdir/UAI/X の記述 (その 3)

```
    window:
      class: smeBSB
      label: Rename
      callback: popup new_name
      window: new_name
        class: dialog
        x: 0
        y: 0
        label: New name?
        value:
        grab: nonexclusive
        window:
          class: command
          label: Ok
          callback:
            error[value]?=!mv @list[value] @new_name[value]
            refresh list
            popdown
          end
        winend
        window:
          class: command
          label: Cancel
          callback: popdown
        winend
      winend
    winend
    window:
      class: smeBSB
      label: Edit
      callback: jterm -e @edit @list[value] &
    winend
    window:
      class: smeBSB
    winend
    window:
      class: smeBSB
      label: Delete
      callback:
        error[value]?=!test -f @list[value] && rm @list[value]
        error[value]?=!test -d @list[value] && rmdir @list[value]
        refresh list
      end
    winend
  winend
winend
```

図 2.14: xdir/UAI/X の記述 (その 4)

```
    window: Help
      class: command
      callback: xterm -e @view @help &
    winend
  winend
  window: working_directory
    class: label
    label: !pwd
  winend
  window:
    class: viewport
    allowvert: true
    window: list
      class: list
      list: !ls @lsoption @rest
      translation: ShortCut
    winend
  winend
winend
```

図 2.15: xdir/UAI/X の記述 (その 5)

Chapter 3

手続き型統合化支援システム: XTSS

XTSS (X Toolkit Service System) は，GUI を定義するために手続き型簡易言語を提供する。XTSS は，"TUI アプリケーション群を統合して GUI アプリケーションを構築する" という UAI/X の思想を継承した統合化支援システムである。XTSS では UAI/X に新たな構想を導入することにより，UAI/X における以下のような問題点を解決する。

- 一行単位で入出力を行なう TUI アプリケーションには対応していたが，エディタのような高度に対話的なアプリケーションを統合できない。
- ウィンドウの組合せの実現を主目的としたため，線分の描画などのグラフィックス命令を提供していない。

XTSS は，X Toolkit レベルの GUI を実現するクライアント/サーバ型のシステムであり，テキスト入出力による GUI プログラミング機能を提供するサーバ XTS (X Toolkit Server) と，複数の TUI アプリケーションを結合する機能を提供するクライアント XTC (X Toolkit Client) という 2 つの独立したシステムで構成される。XTSS は，全ての通信をテキストで行なうという方式を採用することにより，上記の課題を解決できるだけでなく，他にも様々な意義を持つシステムとなっている。

3.1 手続き型統合化支援システムの構想

GUI の外観の生成については，宣言型統合化支援システムである UAI/X において X Toolkit と同等の外観の生成が可能となった。ところが，対話的処理，すなわち，TUI アプリケーションの統合については，テキストエディタのような高度に連続的な TUI アプリケーションへの対応が困難であった。

そこで，まず，対話的処理の中心となるTUIアプリケーション，特に連続型のTUIアプリケーションの統合について考察する。TUIアプリケーションを統合するためには，以下のような通信機構が必要となる。

- 通信路
  複数のTUI間でお互いに情報(データ)を交換するためには，TUIアプリケーション間に通信路が必要となる。通信路の形態としては，ネットワークのトポロジと同様にスター型，ツリー型，バス型，リング型などが考えられる。

- 通信制御機構
  各TUIアプリケーションを修正せずにそのまま利用できるようにするために，どのアプリケーションからデータを読み込むか，読み込んだデータをどのように加工し，どのアプリケーションへ書き込むかなどという入出力制御のための機構が必要となる。

本研究では，通信機構として

- 単純で実装が容易であり，高速に動作すること
- データを転送する際のオーバヘッドが少ないこと
- データの通信制御や加工のために複雑な手続きが必要でないこと

という条件で設計を行ない，以下のような方式を採用した。

- 単純な機構とするため，通信制御機構としては，各アプリケーションの通信相手先や通信条件などの指定のみとし，データの加工はシェル等の外部のTUIアプリケーションに任せることとする。このことにより，データの加工のための処理は，各ユーザが習熟しているシェルを利用すれば良くなるため，加工のために新しい言語を覚える必要もなくなる。
- データ転送のオーバヘッドを少なくするため，パケット化は行なわず，生のデータをそのままやりとりすることとする。
- 通信制御を簡単に行なうため，通信路はスター型とする。スター型では，通信制御を1ヶ所のみで行なうため，データの流れの制御を比較的簡単に実現でき，制御を高速に行なうことも可能である。

次に，外観の生成について考察する。外観の生成のための機能を直接通信機構に組み込むと，単純化して設計した通信機構の役割の範囲を越えることになる。そこで，データの加工部分を外部のTUIアプリケーションに任せたのと同様に，GUIの外観の生成についても外部のTUIアプリケーションで行なうこととする。

以上のような方式に基づく手続き型統合化支援システムXTSSの全体像は以下のようになる。

- 通信機構(XTC)
  データの流れの制御などの必要最小限の通信路および通信制御機能を提供する。
- 主プロセス
  通信機構を利用して複数のTUIアプリケーション間のデータ入出力を制御する際に，中心的な役割を担うTUIプログラムである。他のTUIアプリケーションから読み込んだデータを加工して別のTUIアプリケーションへ転送するなどの役割も果たす。
- GUI化アプリケーション(XTS)
  主プロセス等からの命令によりGUIを生成する機能を提供する。

GUI化アプリケーションであるXTSは，

- X Toolkitと同等のGUI構築が可能であること
- シェル等から命令を記述しやすいように，行単位で命令を記述できること

という条件で設計を行ない，以下のような方式を採用した。

- X Toolkitのライブラリ関数に準拠したXTS命令を提供することにより，X Toolkitと同等のGUI構築を可能とする。このため，基本的にはX Toolkitのライブラリ関数名からXtを除いた名前を持つ命令，Xawのライブラリ関数名と同じ名前を持つ命令などを準備することとなる。
- XTS命令と命令に対するXTS応答はいずれも行単位とする。

3.2 システムの概要

3.2.1 XTSSの全体構成

XTSSは，図3.1に示すようにGUIプログラミング機能を提供するサーバXTSと複数のアプリケーションを結合するための通信路を提供するクライアントXTCから構成される。

XTCは，XTSとその他のアプリケーションをプロセスとして起動し，プロセスが相互にテキストで通信する際の通信路となる。したがって，任意のTUIアプリケーションは，XTSと通信することによってGUIを記述することができ，他の様々なTUIアプリケーションを利用することもできる。

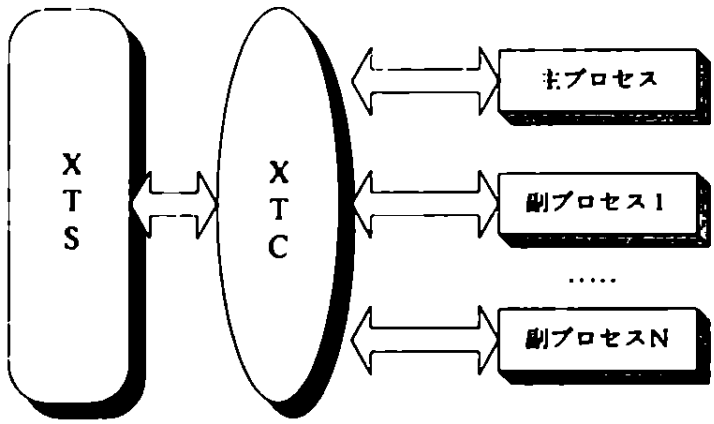

図 3.1: XTSSの構成

3.2.2 XTSとXTS命令

XTSが提供するXTS簡易言語では，X Windowに関する命令をテキストにより指定する。XTSは指定された命令（XTS命令）をX Window上で実行し，実行結果をメッセージ（XTS応答）として出力する。命令も応答も単純なテキストであるため，perl[Wall90]のようにパイプ，ソケットあるいは仮想端末の機能を持つ言語からは直接XTSを起動して使用することができる。それ以外のシェルからはXTCを通信路として使用することによりXTSと通信することができる。

XTS命令は，大きく分けて特殊命令，Xt命令，X命令，Xaw命令の4つに分かれているが，それぞれXTS独自の命令，X Toolkit, X Lib, Athena Widgetに関する命令である。XTS命令の文法(付録A参照)は，

[キー] コマンド名 引数 ... 引数 改行コード

であり，XTS応答の形式は，

[キー] 結果コード メッセージ 改行コード

である。結果コードは，命令が正常に終了すればOK，致命的な異常を起せばFatal，異常終了すればErrorとなる。キーについては，XTS命令で指定したキーがそのままXTS応答

に付加して返される。XTS ではコールバック関数やイベントのアクションにより非同期的に XTS 応答が返される場合があるので，XTS 命令と XTS 応答の同期を取るためにキーを利用する。

XTS にはコマンドの実行機能があるため，通常のコンピュータネットワーク上のサーバのように特定のポートに結合された形式ではなく，各ユーザが XTS を実行ファイルとして直接起動するという形式をとっている。また，XTS ではウィジェットなどの X Window に関する様々の構造にテキストでアクセスするため，表 3.1 に示すようなエントリ・データ型・データ構造の 3 つ組の対応をハッシュ表で管理している。

表 3.1: ハッシュ表の 3 つ組

	エントリ	データ型	データ構造
a	XTS 命令	Function	命令に対する処理関数
b	ウィジェットクラス名	Class	ウィジェットクラス構造体
c	リソースサイズ	Size	データサイズ
d	グラブ名	Int	グラブ種類値
f	アプリケーションコンテクスト名	AppContext	アプリケーションコンテクスト構造体
e	ウィジェット名	Widget	ウィジェット構造体
f	ウィジェット ID	WidgetName	ウィジェット名
g	XTS アクション名	Action	XTS 命令列
h	ディスプレイ名	Display	ディスプレイ構造体
i	グラフィックスコンテクスト名	GC	グラフィックスコンテクスト構造体
j	変数名	Variable	変数値
k	データ名	Data	任意のデータ構造

なお，a〜d のデータはあらかじめ登録されており，その他のものは XTS 命令によって登録される。

3.2.3 XTC と XTC プロセス

XTC は，ある中心となるプロセスが他の TUI アプリケーションとの間で様々な情報をやりとりする際，および，得られた結果を XTS を通じて GUI に反映させる際に，通信路として使用される。XTC を起動する時に複数の TUI アプリケーションを指定すれば，XTC はまず XTS を起動し，次に 1 番目のアプリケーション（主プロセス）を起動し，その後順次他

のアプリケーション（副プロセス）を起動して行く。このようにして起動されたプロセスをXTCプロセスと呼ぶ。起動された各プロセスには以下のような番号（XTCプロセス番号）が割り当てられる。この番号は通信相手を指定するために用いられる。

0: XTS, 1: 主プロセス, 2: 副プロセス, ...

したがって，例えばシェルを主プロセスとして指定すればシェルによるGUIの記述が可能となり，副プロセスとしてTUIアプリケーションを指定することによりシェルでそれらを統合することも可能となる。

3.3 XTSによるウィンドウの操作

3.3.1 特殊命令

特殊命令は，他のXTS命令のようにX Windowに関する処理を行なう命令ではなく，変数の操作，各種モードの設定，アプリケーションの実行，その他補助的な機能を提供する命令である。表3.2に主要な特殊命令を示す。

表 3.2: 主要な特殊命令

CALL	UNIXコマンドを実行する
DATA	位置変数に値を代入する
FORK	UNIXコマンドを子プロセスとして実行する
EVAL	XTS命令を実行し，出力を位置変数に代入する
EXIT	終了モードを参照・設定する
LOCK	ロックモードを参照・設定する
MSG	メッセージ出力モードを参照・設定する
READ	一行読み込み，読み込んだデータを位置変数に代入する
SET	名前付き変数に値を代入する
QUIT	XTSの実行を終了する

なお，位置変数とは，位置で表現される変数であり，数字0, 1, ... が割り当てられる。また，名前付き変数とはAPPCLASSのように名前を持つ変数である。変数を参照する時は，$0, $1, ..., $APPCLASSのように先頭に$を付けて記述する。

XTS命令に対するXTS応答は一度出力バッファに置かれてから出力されるが，XTSは以下のようなモードを持ち，各モードに従ってそれぞれの結果コードに対する処理方法を変える。

- 終了 (EXIT) モード
  結果コードに対する EXIT モードが on の時，実行を終了する。初期値は +fatal であり，致命的なエラーが起きた時に終了するように設定されている。

- ロック (LOCK) モード
  結果コードに対する LOCK モードが on の時，イベントによって実行されるコールバック関数やアクションから非同期的に実行される XTS 命令に対応する XTS 応答をキューに待避する。待避された応答はモードが off になった時に出力される。このモードはキーと同様非同期的な XTS 応答を取り扱うために設けている。初期値は -echo, -fatal, -error, -ok であり，退避は行なわないように設定されている。

- メッセージ出力 (MSG) モード
  結果コードに対する MSG モードが on の時，出力バッファの内容をメッセージとして出力する。初期値は +echo, +fatal, +error, +ok であり，必ずメッセージが出力されるように設定されている。

- 位置変数化モード
  出力バッファの内容を空白で分離し，各部分がそのまま位置変数の値となるように代入操作を行なう。DATA, EVAL 命令によりこのモードになる。

3.3.2 XTS の動作

XTS は図 3.2 に示した処理の流れに従い，以下のような手順で XTS 命令を実行する。

1. XTS 命令を標準入力かファイルから読み込み，ファイルの終わりならば実行を終了する。

2. キー付きかどうかを判定し，キー付ならば出力バッファにキーを複写する。

3. ハッシュ表を参照することにより変数を実際の値に置換しながら，命令と各引数とに分割する。

4. XTS 命令を実行し，XTS 応答を出力バッファに追加する。この際，ハッシュ表を参照して命令に対応する処理関数を呼び出す。X Window 関係の命令に対する処理関数では，ハッシュ表を参照して引数を X Window 用の構造に変換した後，対応する X Window の関数を呼び出す。

5. メッセージ出力モードとXTS応答の結果コードを照合する。出力モードがonの時は，

   (a) 非同期的な応答でかつロックモードがonならば一時的にキューに待避する。

   (b) それ以外の場合は出力バッファの内容を出力する。

6. DATA, EVAL命令ならば，出力バッファの内容を分割し，位置変数に代入する。

7. 終了モードと結果コードを参照し，終了モードがonの時は，実行を終了する。

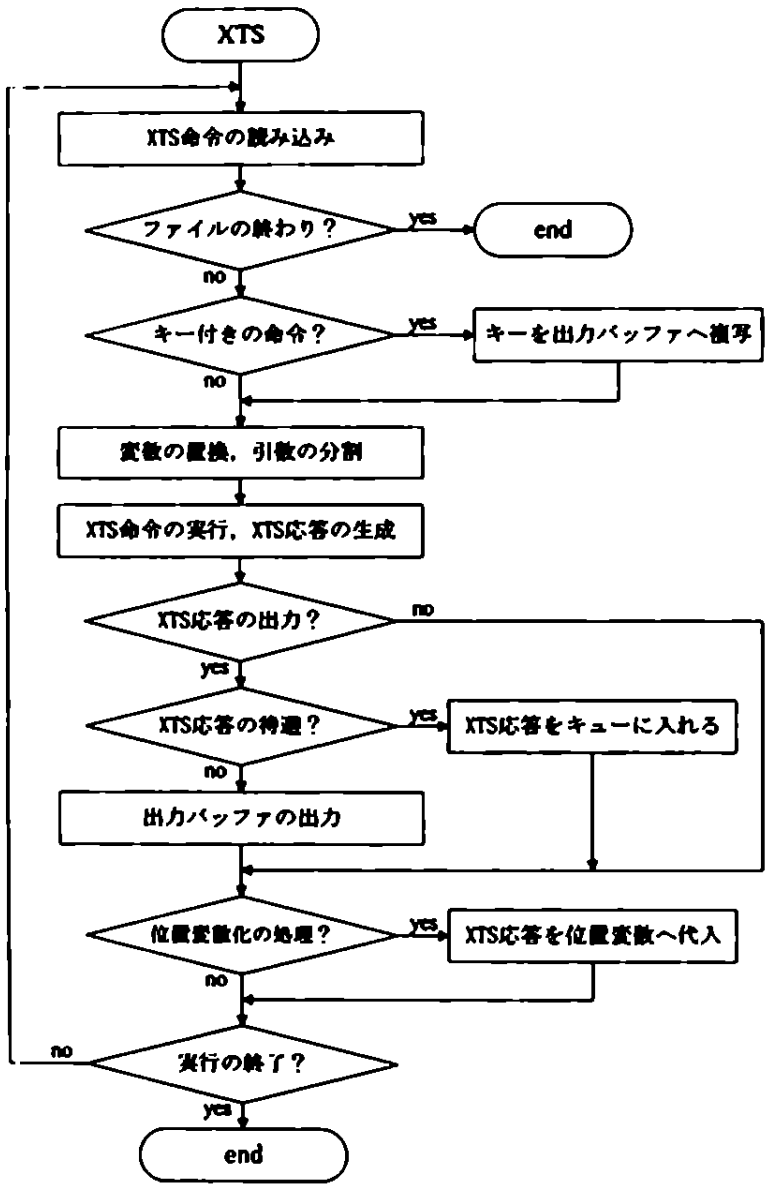

図 3.2: XTSの処理の流れ

3.3.3 GUIアプリケーションの定義

X Windowに関係する中心的な命令はXt命令であり，基本的にはXt関数名からXtを除いた名前を持つが，テキストによる操作に対応できるようになっている。表3.3に主要なXt命令を示す。

表3.3: 主要なXt命令

AppInitialize	アプリケーションコンテクストの初期化およびルートウィンドウの生成を行ない，アプリケーション名とウィジェット名を割り当てる
CreateWidget	ウィンドウを定義し，ウィジェット名を割り当てる
CreatePopupShell	ポップアップウィンドウを定義し，ウィジェット名を割り当てる
Popup	ウィンドウをポップアップする
Popdown	ウィンドウをポップダウンする
RealizeWidget	ウィンドウを初期化する
AppMainLoop	指定されたアプリケーションコンテクストでイベント待ちループに入る
AddCallback	XTS命令を実行するコールバック関数を登録する
AppAddTimeOut	指定された時間に実行するXTS命令を登録する
AppAddAction	XTS命令を実行するアクションを定義し，XTSアクション名を割り当てる
GetValues	リソース値を参照する
SetValues	リソース値を設定する

GUIアプリケーションを定義するには以下のように記述する。

1. AppInitialize命令でX Windowを初期化し，アプリケーションコンテクスト名とルートウィンドウのウィジェット名を設定する。

2. Create(Managed)Widget命令でウィンドウを生成し，ウィジェット名を設定する。さらに子ウィンドウを生成するときは親ウィジェット名として使用する。

3. RealizeWidget命令でウィンドウを初期化する。

4. AppMainLoop命令により，指定したアプリケーションコンテクストでイベント待ちループに入る。

例えば，Hello World! を表示するGUIアプリケーションは図 3.3 のように記述する。

```
AppInitialize appcon Hello   ; ウィジェット Hello の作成
OK 292
CreateManagedWidget 'Hello World!' labelWidgetClass Hello
                             ; ウィジェット Hello World! の作成
OK 2278
RealizeWidget Hello          ; ウィジェット Hello の初期化
OK
AppMainLoop appcon
OK                           ; イベント待ちのループに入る
```


```
QUIT                         ; XTS セッションの終了
```

図 3.3: Hello World!

CreateWidget 命令や SetValues 命令等によりウィンドウの大きさや表示位置等のリソースを設定・変更することもできる。また，CreatePopupShell 命令で定義されたウィンドウを Popup/Popdown 命令でポップアップ／ポップダウンすることでポップアップウィンドウを使用することもできる。

3.3.4 イベントの処理

XTS では，イベントを処理するための方法として，コールバック関数，時間イベント，トランスレーションという以下のような 3 種類の定義が可能である。

- コールバック関数の定義

  コールバック関数として XTS 命令を実行したい場合には，

```
AddCallback <widget> <callback label>
            <data> <XTS instruction> ...
```

により，指定したウィジェット <widget> のコールバック関数としてXTS命令列(<XTS instruction> ...) を登録する。データ指定<data>にはXTS命令列を実行する際の位置変数を指定する。データ指定は，NULL以外が指定されれば，XTS命令として一度実行され，実行結果として得られた位置変数がXTS命令列を実行する際の位置変数となる。したがって，文字列を渡す場合はDATA命令，XTS命令の実行結果を渡す場合はEVAL命令を利用するが，この時のXTS応答は出力しない。コールバックラベルで指定されたイベントが起きる度毎に，イベントが起こったウィジェット名を $0 に，データ指定を $1 以降に代入して，XTS命令列が実行される。例えば，yes/noを表示するためのボタンを持つGUIアプリケーションは図3.4のようになる。

```
AppInitialize appcon YesNo -geometry +0+0 NULL
CreateManagedWidget box compositeWidgetClass YesNo \
        width 110 height 55
CreateManagedWidget lable labelWidgetClass box \
        label 'Are you OK ?' x 10 y 5
CreateManagedWidget yes commandWidgetClass box x 20 y 30
CreateManagedWidget no commandWidgetClass box x 60 y 30
AddCallback yes callback NULL 'ECHO $0'
                            ; yes を表示する callback
AddCallback no callback NULL 'ECHO $0'
                            ; no を表示する callback
RealizeWidget YesNo
AppMainLoop appcon
```

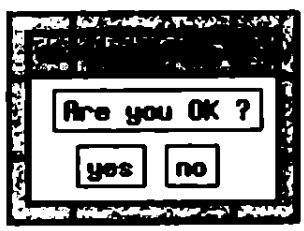

図 3.4: yes/no ボタン

- 時間イベントの定義

  ある時間後あるいは時間毎にXTS命令を実行したい場合には，

  ```
  AppAddTimeOut <appcontext> <time>
  ```

```
<data> <XTS instruction> ...
```

により，<time>ミリ秒後に実行するXTS命令を登録する。+<time>と指定した時は<time>ミリ秒毎に繰り返し実行される。データ指定，XTS命令についてはAddCallback命令と同様である。(図3.6参照)

- トランスレーションの定義
  トランスレーションの定義は，AppInitialize命令を用い，以下のようなフォールバックリソースを記述することにより行なう。

```
"*translations: トランスレーションの指定"
```

トランスレーションの指定はX Windowのリソース指定に準じて記述するが，アクションとしてXTS命令を呼び出したい時には，

```
AppAddAction <appcontext>
             <XTS action> <XTS instruction> ...
```

によって呼び出したいXTS命令列をXTSアクションとして登録し，トランスレーションの指定で，

```
XTS(XTSアクション名 引数 ...)
```

のように呼び出す。XTSアクションが呼ばれた際には，アクションが起こったウィジェット名を$0に，アクションの引数を$1以降に代入し，XTS命令列が実行される。例えば，3つのマウスボタンのうちどのボタンが押されたかを表示するGUIアプリケーションは図3.5のように定義できる。

```
AppInitialize appcon Button NULL \
        '*message.translations: #override \
                <Btn1Down>: XTS(button Left) \\n\
                <Btn2Down>: XTS(button Center) \\n\
                <Btn3Down>: XTS(button Right)'
CreateManagedWidget paned panedWidgetClass Button
CreateManagedWidget message labelWidgetClass paned \
        label 'Push Here'
```

```
CreateManagedWidget result labelWidgetClass paned
AppAddAction appcon button \
        'SetValues result label $1'
RealizeWidget Button
AppMainLoop appcon
```

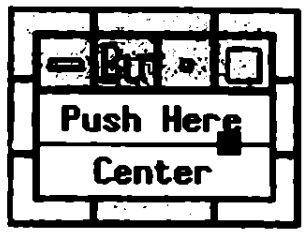

図 3.5: XTSアクションの指定

3.3.5 アプリケーションの呼び出し

TUIアプリケーションの出力を取り出すCALL命令とリソース値を参照・設定するGet-Values, SetValues命令を組み合わせることによって単純なTUIアプリケーションを統合することができる。例えば，現在時刻を表示するGUIアプリケーションは図3.6のようになる。ただし，$* は $1 以降の全ての位置変数を表わす特殊な変数である。

```
AppInitialize appcon Date
CreateManagedWidget clock labelWidgetClass Date \
        label '  D i g i t a l  C L O C K  '
AppAddTimeOut NULL +1000 NULL \
        'EVAL CALL date' \          ; date コマンド実行の登録
        'SetValues clock label "$*"'
RealizeWidget Date
AppMainLoop appcon
```

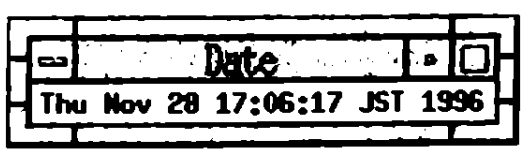

図 3.6: dateコマンドアプリケーション

3.4 XTCによるアプリケーションの統合

3.4.1 XTCコマンド

XTCによって起動されるXTCプロセスには，メッセージを送る相手プロセスである送信先と送信の方法を決定する送信モードがあり，表3.4に示すXTCコマンドにより設定する。

表3.4: XTCコマンド一覧

put 数字	数字で指定したXTCプロセスを送信先とする
get 数字	数字で指定したXTCプロセスの送信先を現プロセスにする
数字	以後の通信を数字で指定したXTCプロセスとの間で行なう。put 数字, get 数字の2つの命令と等価である
数字 メッセージ	数字で指定したXTCプロセスを送信先とし，そのプロセスの送信先となった後，メッセージを送る
(no)line	通信の単位を行単位(line mode)にするか文字単位(no-line mode)にするかを指定する
(no)listen	他のプロセスからの通信を受け取る(listen mode)か無視する(nolisten mode)かを指定する
(no)passive	XTCコマンドを解釈しない(passive mode)か解釈する(nopassive mode)かを指定する
(no)prefix	: を付けたときのみプロセス間通信を行なう(prefix mode)か標準出力に出力したいときに : を付与する(noprefix mode)かを指定する
(no)wait	通信を行なった後相手の応答を待つ(wait mode)か待たない(nowait mode)かを指定する

なお，送信先は，特に指定のない場合には，

XTS -> 主プロセス，主プロセス -> XTS，副プロセス -> 主プロセス

となっている。

3.4.2 XTCの動作

XTCの動作は，以下のような手順となる。

1. XTCプロセスをプロセス番号0, 1, ... の順に走査し，出力メッセージがあるかどうかを調べる。

2. そのプロセスがlineモードならば一行単位で，そうでなければ文字単位でメッセージを読み込む。

   (a) passiveモードならばメッセージを通信先に設定する。

   (b) prefixモードでメッセージが：で始まるか，noprefixモードで：以外で始まる場合，メッセージの先頭が英大文字ならばメッセージを通信先へ設定し，小文字ならばXTCコマンドとして実行する。

   (c) それ以外の場合は，メッセージを標準出力へ出力する。

3. 通信先がlistenモードならば設定されたメッセージを送信する。

4. waitモードならば通信先からの応答を待つ。

3.4.3 perlによる記述

perlは独自のソケット機能を持つので，XTCを使用せずにXTSとやりとりすることも可能である。このため，perlのライブラリとして以下の関数を提供し，記述をより容易にしている。

- &XTSopen(ホスト名, ユーザ名): “ホスト名”の“ユーザ名”としてXTSを起動し，以後のXTS命令を有効にする
- &XTSfile(ファイル名1, ファイル名2, ...): ファイル名1, ファイル名2, ... からXTS命令を読み込み，XTSへ送る
- &XTSexec(文字列1, 文字列2, ...): 文字列1, 文字列2, ... をXTSに送り，応答を無視する
- &XTStalk(文字列): 文字列をXTSに送り，応答を無視する
- &XTSwrite(文字列): 文字列をXTSに送る
- &XTSread(): XTSから応答を受ける
- &XTSclose(): XTSとの通信を閉じる

3.4.4 XTCによるアプリケーションの統合

1.4.2 節で述べたTUIアプリケーションの型それぞれについて，統合方法をまとめておく。

単純型 AddCallback 命令で登録するXTS命令列の中に，アプリケーションを呼び出すCALL命令を指定することで実現できる。

繰り返し型 AppAddTimeOut 命令で登録するXTS命令列のなかにCALL命令を指定する(図 3.6 参照)か，XTSアクションを定義するAppAddAction命令で登録するXTS命令列の中にCALL命令を指定すればよい。

連続型 XTCを利用して統合する。図 3.7 に行単位連続型の例として，計算プログラムbcを統合する例を示す。文字単位連続型の統合はXTCをnolineモードにすることで達成できるが，具体例については次節で述べる。

```
Initialize bc BC
CreateManagedWidget paned panedWidgetClass bc
CreateManagedWidget dialog dialogWidgetClass paned \
        label 'Enter expression ?' value ''
CreateManagedWidget answer labelWidgetClass paned
XawDialogAddButton dialog 'Do Calc' NULL \
        'EVAL XawDialogGetValueString dialog' \
        'ECHO O$1' \
        READ \
        'SetValues answer label $0'
XawDialogAddButton dialog Quit NULL QUIT
RealizeWidget bc
MainLoop
EXEC 'MSG -ok' 'ECHO OK'

% xtc -fbc.cat -ipassive bc
```

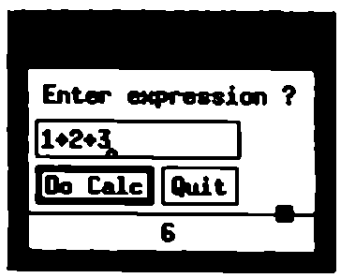

図 3.7: 行単位連続型の例

3.5 XTSSの応用

3.5.1 動的なGUIの定義

perlによる動的なGUI定義の例として，現在稼働中のホスト名をruptimeコマンドで獲得し，得られたホスト名をラベルに持つボタンを動的に生成し，ボタンが押されれば対応するホストにログインするというGUIアプリケーションhostsを図3.8に示す。

```
#!/usr/local/bin/perl
require 'xtc.pl';
&ITSopen();
&ITSexec('Initialize hosts Hosts',
    'CreateManagedWidget paned panedWidgetClass hosts width 500',
    'CreateManagedWidget Quit commandWidgetClass paned',
    'AddCallback Quit callback NULL QUIT',
    'CreateManagedWidget box boxWidgetClass paned');
open(RUPTIME, "ruptime|") ||
    die "Coundn't execute ruptime command: $!\n";
while(<RUPTIME>) {
    split;
    &ITSexec("CreateManagedWidget @_[0] commandWidgetClass box",
             "AddCallback @_[0] callback NULL
                 'FORK xterm -e rlogin @_[0]'")
        if @_[1] eq 'up';
}
&ITSexec('RealizeWidget hosts', 'MainLoop', 'MSG -ok');
```

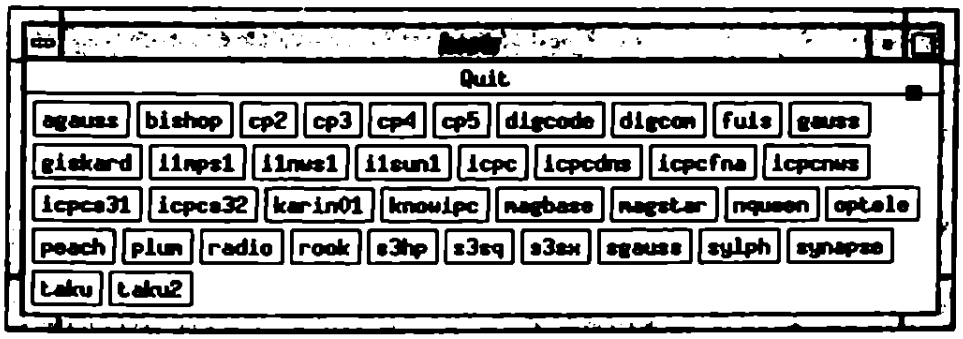

図 3.8: 動的なGUI定義の例

ruptimeコマンドによって得られるホスト情報から必要なホスト名だけを切り出し，ウィンドウを動的に生成するという動作をperlで簡単に記述できている。もちろん，perl以外

のシェルを使用して記述することも同様に可能であり，言語を選ばないことが分かる。

3.5.2 連続型アプリケーションの統合

高度な文字連続型アプリケーションの例として，エディタ vi を対象とし，X11R5 のクライアントとして提供されている xedit と同様の機能を有するメニューを持つGUIアプリケーション vi/XTSS を取り上げる。(図 3.9 参照)

```
% xtc -fvi.cat -inoline,nowait,2,passive vi -inoline,nowait,passive -
```

```
Initialize vi VI
CreateManagedWidget paned panedWidgetClass vi
#
# File Operations
#
CreateManagedWidget file panedWidgetClass paned
#
# file commands
#
CreateManagedWidget Quit commandWidgetClass file
CreateManagedWidget FileName dialogWidgetClass file
        label 'File Name' value ''
CreateManagedWidget Load commandWidgetClass FileNam
CreateManagedWidget Save commandWidgetClass FileNam
CreateManagedWidget NoSave commandWidgetClass FileN
AddCallback Quit callback NULL \
        'ECHO ^[:q'
AddCallback Save callback NULL \
        'ECHO ^[:wq!'
AddCallback Load callback NULL \
        'EVAL XawDialogGetValueString FileName' \
        'ECHO ^[:vi $1'
AddCallback NoSave callback NULL \
/ECHO
```

vi/XTSS

Quit

File Name

vi.mtg

Load | Save | NoSave

Search for

ECHO

Forward | Backward

Replace with

Replace | Replace All

図 3.9: vi/XTSS のメニュー

xedit は大きく分けてファイル操作のためのトップメニュー領域とファイルの編集を行なう編集領域という 2 つの領域に分かれる。トップメニュー領域には，終了のための Quit ボタン，ファイルを読み込む Load ボタン，ファイルを保存する Save ボタン，ファイル名を指定するダイアログウィンドウが配置されている。編集領域では，'S あるいは 'R を入力することによりサーチ/置換ウィンドウがポップアップされる。ここにはサーチする文字および置換対象の文字を指定するダイアログウィンドウと前方サーチ (Forward)/後方サーチ (Backward)/置換 (Replace)/全ての置換 (Replace All) という 4 つのボタンが配置されている。

XTSS では vi を利用して同様の機能を実現するが，現在端末と同じ機能を持つ独立したウィジェットが提供されていないので，メニュー領域と編集領域からなる 1 つのウィンドウを構成することはできない。そこで，メニュー領域を編集領域から分離し，さらに，トップメニューとサーチ／置換ウィンドウを 1 つにまとめて表示することにした。

vi/XTSS では，図 3.10,3.11 のようなメニューウィンドウの定義を XTS に送った後，vi を主プロセス，標準入出力すなわち端末を副プロセスとして noline, nowait, passive モードで起動する。このため vi を通常のように端末から使用できるだけでなく，メニューウィンドウからの出力も vi への入力となる。例えば，Load ボタンは，ダイアログウィンドウからファイル名を得て「:vi ファイル名」を vi へ送ることで実現しており，前方サーチのための Forward ボタンは，サーチ文字列を得て「/サーチ文字列」を送ることで実現している。なお，vi をコマンド入力モードに移行させるために，vi へ送る文字列の先頭にエスケープコードを付加している。

3.6 評価

TUI アプリケーション間の通信機能を実現する XTC を提供することで TUI アプリケーションを比較的簡単に統合できることを示した。また，XTC は最小限の通信制御機能しか提供しないが，シェルを主プロセスとして統合することで他の TUI アプリケーションからのデータを加工したり XTS を利用して GUI の外観を生成できることも示した。

XTSS の特徴として，以下のような点が挙げられる。

- TUI アプリケーションの統合

  先に述べたように，一般にプログラミング言語から TUI アプリケーションを呼び出すには，プロセス間通信などの特殊な知識が必要となる。これに対して，XTSS を利用すれば，アプリケーションとしては TUI 部までを記述した TUI アプリケーションを作成し，XTS により GUI を実現し，それらを XTC で 1 つのシステムとして統合することができる。

  XTS および XTC を用いてテキストエディタ vi にメニューを付加するなどの TUI アプリケーションの統合が比較的簡単に実現できることは既に述べた通りである。

- GUI 定義に要する記述量の少なさ

  XTS では X Toolkit の関数に似た命令を提供するだけでなく，様々な特殊命令も提供しており，これらの命令と XTC が提供する通信に関する様々な機能を組み合わせるこ

とにより，GUI の記述が比較的簡単になる。前節で取り上げた GUI アプリケーションを C 言語で直接記述した場合に対する記述量の比較を表 3.5 に示す。XTSS では，記述量が 1/2〜1/3 になり記述が簡単化されることが分かる。

表 3.5: 記述量の比較

システム	hosts	hosts/XTSS	vi	vi/XTSS
言語	C	XTS+perl	C	XTS+XTC
行数	60	16	236	64
バイト数	127	61	597	216

- 対話的な GUI 記述環境

  XTS は UAI/X と同様にインタプリタ系の言語であり，命令が解析あるいは実行される際に文法エラーや実行時エラーの表示を行なうため，C 言語などのコンパイラ系の言語と比較してデバックが非常に容易である。

また，先に提案した UAI/X で課題として残されたもののうち，高度な対話的アプリケーションの統合については XTC の機能により，グラフィックス命令の提供については XTS が X 命令を提供することで解決する。

XTSS ではさらに，UAI/X では実現できなかった以下のような重要な特徴を備えていることも示した。

- 言語を選ばない GUI の記述

  一般に，GUI アプリケーションは，GUI 用に拡張されているか，あるいは，GUI 用のライブラリを使用できるプログラミング言語あるいは GUI 専用の言語を用いて記述する必要があった。XTSS では XTS 命令をテキストで指定できるため，テキストの入出力が可能な言語であればどのような言語からでも GUI プログラムを記述でき，特別のライブラリ等は不要である。したがって，本格的なプログラミング言語ではなくシェルなどを用いるユーザでも GUI アプリケーションを記述することができる。

- 動的な GUI 定義

  UAI/X では使用する全てのウィンドウの階層やクラス等の情報を記述しておかねばならないため静的な GUI 定義しかできなかった。しかし，XTSS では必要に応じてウィンドウを生成したり削除したりできるので，UAI/X より動的に GUI を変更できる。

```
Initialize vi VI
CreateManagedWidget paned panedWidgetClass vi
#
# File Operations
#
CreateManagedWidget file panedWidgetClass paned
#
# file commands
#
CreateManagedWidget Quit commandWidgetClass file
CreateManagedWidget FileName dialogWidgetClass file \
        label 'File Name' value ''
CreateManagedWidget Load commandWidgetClass FileName
CreateManagedWidget Save commandWidgetClass FileName
CreateManagedWidget NoSave commandWidgetClass FileName
AddCallback Quit callback NULL \
        'ECHO ^[:q'
AddCallback Save callback NULL \
        'ECHO ^[:wq!'
AddCallback Load callback NULL \
        'EVAL XawDialogGetValueString FileName' \
        'ECHO ^[:vi $1'
AddCallback NoSave callback NULL \
        'ECHO ^[:vi!'
#
# Search Operations
#
CreateManagedWidget search boxWidgetClass paned
#
# Search commands
#
CreateManagedWidget find dialogWidgetClass search \
        label 'Search for' value ''
CreateManagedWidget Forward commandWidgetClass find
AddCallback Forward callback NULL \
        'EVAL XawDialogGetValueString find' \
        'ECHO ^[/$1'
CreateManagedWidget Backward commandWidgetClass find
AddCallback Backward callback NULL \
        'EVAL XawDialogGetValueString find' \
        'ECHO ^[?$1'
```

図 3.10: vi/XTSS の記述 (その 1)

```
#
# Replace commands
#
CreateManagedWidget replace dialogWidgetClass search \
        label 'Replace with' value ''
CreateManagedWidget Replace commandWidgetClass replace
AddCallback Replace callback NULL \
        'EVAL XawDialogGetValueString find' \
        'SET FIND "$*"' \
        'ECHO ^[/$FIND' \
        'EVAL XawDialogGetValueString replace' \
        'ECHO ^[:s/$FIND/$1/'
CreateManagedWidget 'Replace All' commandWidgetClass replace
AddCallback 'Replace All' callback NULL \
        'EVAL XawDialogGetValueString find' \
        'SET FIND "$*"' \
        'EVAL XawDialogGetValueString replace' \
        'ECHO ^[:%s/$FIND/$1/g'
RealizeWidget vi
MainLoop
EXEC 'MSG -ok' 'ECHO OK'
```

図 3.11: vi/XTSS の記述 (その 2)

Chapter 4

視覚型統合化支援システム: XTSS Builder

UAI/X および XTSS は，GUI の記述を簡単化するために簡易言語を提供しているが，たとえ簡易言語を用いたとしても，実際に GUI アプリケーションを記述することは，特に初めて利用するユーザにとってはかなり困難であり，新しい言語を学習しなければならないという抵抗もある。このため，GUI の構築自体も GUI 環境下で実現する GUI ビルダの重要性が認識されるようになってきている。GUI ビルダである視覚型統合化支援システム XTSS Builder は，Creator と Analyzer から構成される。Creator は，視覚的操作で GUI の構成を指定し，簡易言語の記述を生成する機能を提供する。Analyzer は既存の GUI アプリケーションを参照してそのアプリケーションを記述するための簡易言語を自動生成する機能を提供する。

4.1 視覚型統合化支援システムの構想

視覚型統合化支援システムでは，ユーザとの対話的処理の部分までを支援することは困難なため，この部分については XTSS に任せ，GUI の外観の生成に関する部分のみを支援する。したがって，システムとしては，視覚的な操作で GUI の構成を指定する環境を提供し，GUI の記述，すなわち XTS 簡易言語を生成することになる。この機能を提供するシステム部分が Creator である。

一般に GUI ビルダにおける GUI の構成の指定方法は以下のように大別できる。

1. 視覚的に表示された様々な部品から適切な部品を選択し白紙のウィンドウに張り付ける方法

2. メニュー等から部品名を選択し，その親子関係や位置関係を指定する方法

1.の方がより視覚的な操作であるが，2.に比べて一般に実現が困難である。ここでは，できる限り操作が単純で，実現が比較的容易で，しかも充分に視覚的な操作を提供することを目標としたため，基本的には2.の方法を選択しているが，メニューではなくツリーによる表示・操作を全面的に採用する。すなわち，GUI用の部品を部品のクラス階層のツリー構造に配置しておき，そこから選択した部品を白紙のツリーに親子関係を考えながら張り付けるという指定方法を取る。各部品の属性については，部品の持つ属性の一覧表から選んでその値を指定する。XTSSの立場から考えると，視覚型統合化支援システムは，記述能力は高いがUAI/Xに比べてやや記述が難しくなっているXTSSの記述を視覚的に援助するシステムであるとみなすこともできる。

さらに，通常のGUIビルダでは提供されない機能として，XTSSあるいはUAI/X以外で記述された既存のGUIアプリケーションを統合化支援システムで再構築する作業を援助するために，GUIアプリケーションの部品の親子関係を自動的に取得し，簡易言語の記述に変換する機能の提供も行なう。これにより，他の方法で記述されたGUIアプリケーションをXTSSあるいはUAI/Xによる記述に乗り換えることも容易になると期待される。なお，この機能を提供するシステム部分がAnalyzerである。

4.2 システムの概要

図 4.1 が XTSS Builder のトップメニューであり，マウス操作によって Creator の起動，Analyzer の起動，ビルダの終了 (Quit) を選択することができる。

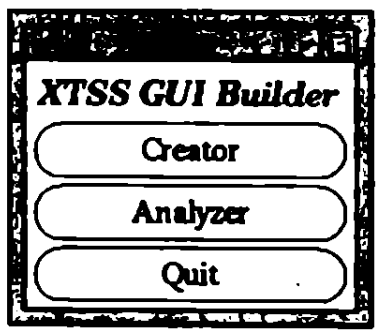

図 4.1: XTSS Builder トップメニュー

一般にユーザが XTSS Builder を使用する状況について考察すると，次のような場合が

ある。

1. 新たに独自のGUIアプリケーションをXTSSによって作成する。

2. XTSSで記述された既存のGUIアプリケーション(XTSSアプリケーション)を修正する。

3. 既存のGUIアプリケーションを参考にしてXTSSアプリケーションを作成する。

1. の場合は，Creatorにより全く新規にGUIアプリケーションを構築し，XTS簡易言語(プログラム)へ変換し，ファイルに保存する(save)ことができる。2. の場合も，CreatorへXTSプログラムを読み込み(load)，GUIアプリケーションを修正し，再度保存し直せばよい。3. の場合には，まず，Analyzerによって既存のGUIアプリケーションをXTS簡易言語に変換しファイルへ出力する(dump)。次にCreatorを起動し，ファイルを読み込めば，視覚的にGUIアプリケーションを改良して行くことができる。この様子を図4.2に示す。

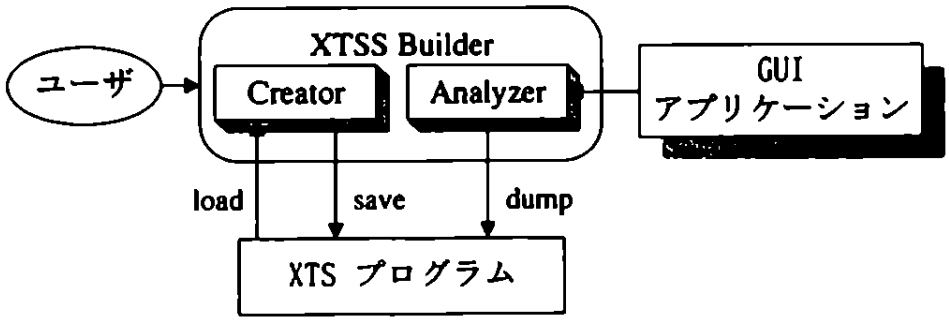

図 4.2: XTSS Builderの全体構成

4.3 Creator

4.3.1 システムの概要

Creatorは，視覚的な操作によるXTSSアプリケーションの構築を支援する。X Toolkitを利用したGUIの構築ではウィジェットの親子関係とそのリソース値を指定することによりGUIが決定されるため，GUIビルダの操作性は親子関係およびリソース値の指定方法によって決まる。先にも述べたように，最も視覚的な操作方法は，視覚的な部品として用意された各ウィジェットクラスの中から，ユーザがクラスを部品として選択し，マウスによりその

部品を置く位置を指定すると，システムが実際にウィジェットを生成しそれを配置することにより全体のGUIを決定する方法である。しかし，この方法を実用的なものにするにはかなり大きなシステムが必要である。しかもそのようなシステムを構築したとしても，例えばリソース値の指定をマウス操作だけで行なうことは無理があるため，視覚的な操作だけで全てを行なうのは所詮不可能である。そこで本システムでは，かなり視覚的な操作性に富み，しかも実現が比較的簡単な方法を採用した。

Creator では，ユーザは Athena Widget クラスのツリー構造からマウスでクラスを選択し，ユーザが構築するGUIのツリー構造の中に設定して行くことによりGUIアプリケーションを構築する。クラスの選択もGUIアプリケーション中のウィジェットの親子関係の設定もツリー構造上で行なうため，統一的な操作性が保たれている。リソース値の設定は，各ウィジェットのリソース名の一覧からリソース名を選択し，それに対するリソース値を入力するという方法をとる。また，この方法では残念ながら作成中のGUIアプリケーションの形状は見えないので，作成途中の形状を表示する機能も提供している。図 4.3 に Creator の外観を示す。

Creatorは，以下の3つのウィンドウから構成される。

メインパネル　操作指定領域，コマンド領域，メッセージ領域，配置領域の4つの領域から構成される。操作指定領域では，ウィジェットの配置 (put widget)，リソースの設定 (resource box)，ウィジェット名の設定 (set name)，ウィジェット名を表示するかクラス名を表示するかの選択 (widget name/class name)，ウィジェット部分木の移動 (move part)，ウィジェット部分木の削除 (del part)，ウィジェットの削除 (del one)，ウィジェットの前方への挿入 (ins ahead)，ウィジェットの上方への挿入 (ins upside) の中から操作を選択する。コマンド領域では，ツリーの初期化 (clear all)，GUI全体の表示 (show)，ウィジェット部分木のGUIの表示 (show part)，XTSプログラムの読み込み (load file)，XTSプログラムの格納 (save file)，Creator の終了 (quit) という常に使用できるコマンド機能を提供する。メッセージ領域では，エラーメッセージなどの様々な情報を表示する。配置領域では，ツリー構造の中にウィジェットを配置することにより，構築しようとするGUIアプリケーションを定義する。

ウィジェット ボックス　Athena Widget のクラスの階層関係をツリー構造で表現したウィンドウ。ユーザは適切なクラスをマウスにより選択することができる。

リソースボックス　コマンド領域，入力領域，リソース名リスト表示領域の3つの領域か

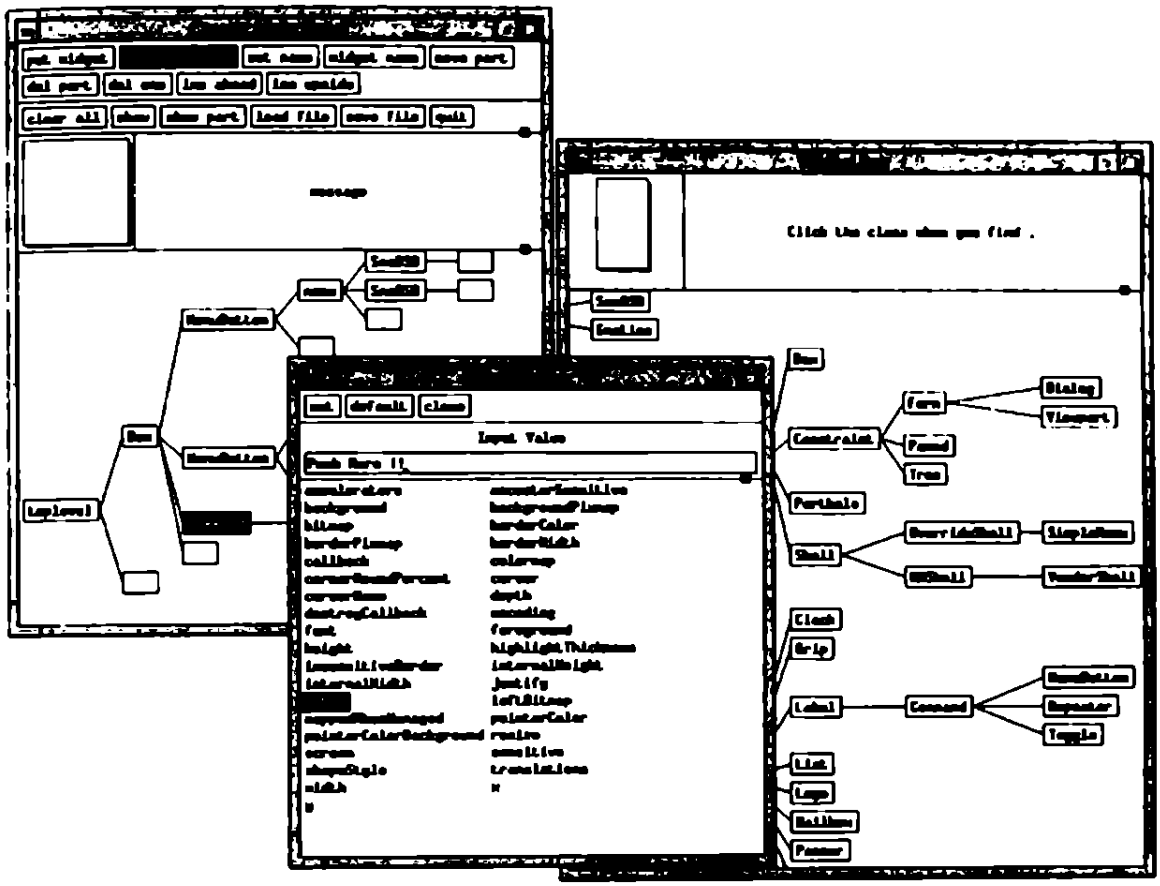

図 4.3: Creator

ら構成されている。コマンド領域では，リソース値の設定 (set)，リソース値を暗黙の値に戻す (default)，すなわちリソース値を削除する，およびリソースボックスの終了 (close) というコマンド機能を提供する。入力領域では，リソース値の入力を行なう。リソース名リスト表示領域では，設定可能なリソースを表示し，値を設定しようとするリソース名を選択する。

操作方法について以下に簡単に触れておく。

GUIアプリケーションにおけるウィジェットの親子関係を指定するためには，まずGUIアプリケーションで使用するウィジェットのクラスをウィジェットボックスから選択し，次に，メインパネルの操作指定領域で put widget を指定する。そして，配置領域中のツリー構造上の適切なノードをクリックすることにより指定したクラスのウィジェットを配置できる。

ツリー構造上で配置可能な位置には自動的に空白の長方形が表示されるため，ユーザはスムーズにウィジェットを配置できるようになっている。

リソースを設定するには，操作指定領域で resource box を指定する。この後，配置領域のノードをクリックすれば，リソースボックスがポップアップされる。設定したいリソース名をリソース名リストから選択し，入力領域にマウスを合わせ，リソース値をキーボードから入力し，set ボタンをクリックすればよい。

ウィジェットの部分木を移動する場合には，操作指定領域で move part を指定し，移動元と移動先をクリックする。削除する場合には，操作指定領域で del one あるいは del part を指定し，削除するウィジェットあるいは部分木の頂点をクリックする。挿入する場合には，ins ahead あるいは ins upside を指定し，基準となるウィジェットをクリックすると，その前方あるいは上方に空白の長方形が追加されるので，put widget によりウィジェットのクラスを指定すればよい。

XTS プログラムを Creator に読み込むためには，メインパネルのコマンド領域の load file をクリックし，XTS プログラムを生成するには，GUI を構築した後 save file をクリックすればよい。いずれの場合もダイアログウィンドウがポップアップされるので，ファイル名を入力すれば XTS プログラムの読み込みあるいは書き込みを行なうことができる。現状の GUI 全体を確認するには，メインパネルのコマンド領域で show をクリックする。一部を確認したいときは，show part を指定し，確認したいウィジェットの頂点をクリックすればよい。

4.3.2 XTS プログラムの解析・生成

Creator は，内部のデータ構造として，各ウィジェットの情報用に WInfo 構造体，リソース情報用に ResourceCell 構造体を使用している。

WInfo 構造体は，ツリーのノード間の関係を表すために，ノードを表現しているウィジェットを指す widget，ツリー構造上の親の WInfo を指す parent，子の WInfo を指す child，兄弟を指す brother という要素を持ち，そのノードのリソースの状態を示す ResourceCell 構造体を指すために resource という要素を持つ。その他に，ウィジェットの名前を示す name，クラス名を示す class，クラスの番号を示す class_num，生成されるウィジェットの種類を示す type，付加的な情報を保持する etc がある。type としては，INITIALIZE, APPINITIALIZE, CREATE, MANAGE, SHELL, POPUP があり，それぞれ Xt 命令の Initialize, AppInitialize, CreateWidget, CreateManagedWidget, CreateShell, CreatePopupShell というウィジェット生成用命令に対応する。

ResourceCell 構造体は，リソース名を示す resource_name，リソース値を示す resource_value，

次の ResourceCell 構造体を指す next という要素を持つ。

Creator による XTS プログラムの解析は，以下のような手順で行なわれる。

1. CreateManagedWidget 等のウィジェット生成用命令の場合には，
   (a) 新しい WInfo 構造体を生成する。命令に対する引数を解析することにより，name, class, type を求め，クラス名 class からクラス番号 class_num を求めてそれぞれの値を WInfo に設定する。
   (b) 親の名前をキーにして親の WInfo を検索し，取り出した親の WInfo との間で親子関係のリンクを張る。
   (c) ウィジェット名をキーにして WInfo を保存する。
   (d) リソースが指定されている場合には，リソース名を resource_name に，リソース値を resource_value に設定した新しい ResourceCell 構造体を生成し，構造体間を next により結合する。
2. それ以外の命令は，一行単位に単純な文字列として直近の WInfo の etc に保存する。

Creator による XTS プログラムの生成は，以下のような手順で行なう。

1. ウィジェット階層のトップの WInfo の name, class, resource を取り出し，type に応じて

   ```
   Initialize <name> <class> NULL <resource>
   ```

   あるいは

   ```
   AppInitialize appcon <class> NULL NULL <resource>
   ```

   を出力する。ただし，<resource>は，ResourceCell 構造体の要素である resource_name と resource_value を

   ```
   <resource_name> <resource_value> ....
   ```

   のように順に出力したものである。
2. etc に情報が保存されている場合には，その情報を出力する。
3. 子あるいは兄弟の WInfo がある場合には，まず子について，次に兄弟について，その WInfo の name と class を取り出し，type に応じて，

```
Create(Maneged)Widget <name> <class> <parent> <resource>
CreatePopupShell <name> <class> <parent> <resource>
```

あるいは

```
AppCreateShell <name> <appclass> <class> <parent> <resource>
```

を出力する。ただし，<parent> には親の WInfo のウィジェット名を用い，<appclass> には <name> と同じものを用いている。

4. 全ての子および兄弟に対して同様の処理を再帰的に行なう。

5. 全ての子および兄弟を生成したら，トップの WInfo の name, type を参照し，

```
RealizeWidget <トップの name>
MainLoop あるいは AppMainLoop appcon
```

を出力する。

なお，show ボタン等による GUI の表示は，上記のようにして XTS プログラムを生成し，ファイルに格納し，そのファイルに対して，

```
xtc -f ファイル名
```

を実行することにより実現している。

図 4.4 に図 4.3 のツリーから生成された XTS プログラムとその表示結果を示す。

```
Initialize toplevel Toplevel
CreateManagedWidget Box boxWidgetClass toplevel width 300
CreateManagedWidget MenuButton menuButtonWidgetClass Box
CreatePopupShell menu simpleMenuWidgetClass MenuButton
CreateManagedWidget SmeBSB smeBSBObjectClass menu
CreateManagedWidget SmeBSB smeBSBObjectClass menu
CreateManagedWidget MenuButton menuButtonWidgetClass Box
CreatePopupShell menu simpleMenuWidgetClass MenuButton
CreateManagedWidget SmeBSB smeBSBObjectClass menu
CreateManagedWidget SmeLine smeLineObjectClass menu
```

```
CreateManagedWidget SmeBSB smeBSBObjectClass menu
CreateManagedWidget Command commandWidgetClass Box label "Push Here !!"
RealizeWidget toplevel
MainLoop
```

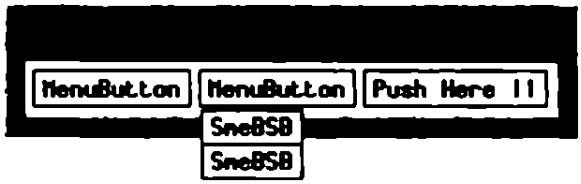

図 4.4: 生成された XTS プログラムと表示結果

4.4 Analyzer

4.4.1 システムの概要

Analyzer は，既存の GUI アプリケーションの GUI 部をそのまま，あるいは多少改良して利用したい場合に使用する。Analyzer の機能を実現するには，利用したい GUI アプリケーションを起動しておき，そのアプリケーションをマウスで指定することによりウィジェット情報を取得し，簡易言語を生成するという手順になるが，このためには以下のような動作が必要となる。

1. 対象となる GUI アプリケーションに Analyzer 側から，ウィジェット情報の取得を依頼するイベントを送る。

2. GUI アプリケーションがそのイベントを理解し対応する処理を行ない，Analyzer にウィジェット情報を送り返す。

すなわち，他の GUI アプリケーションのウィジェット情報を取り出すためには，GUI アプリケーションがイベントを理解できるようにアプリケーションにイベントハンドラを登録しておかなければならないのである。しかし一般に目的のアプリケーションは不特定なので，その都度イベントハンドラを登録することは非常に困難である。ところが，Athena Widget には editres(X Toolkit アプリケーション用ダイナミックリソースエディタ) 用のイベントハンドラが既に登録されており，editres はこの editres 用プロトコルを用いてクライアント間通信を行ない，ウィジェット情報の取得を行なっている。このため Analyzer は editres を利用

し，editresのコマンドメニューに，XTSおよびUAI/Xプログラム生成用のメニューを追加するという形で作成した。図 4.5 がAnalyzerのトップウィンドウである。

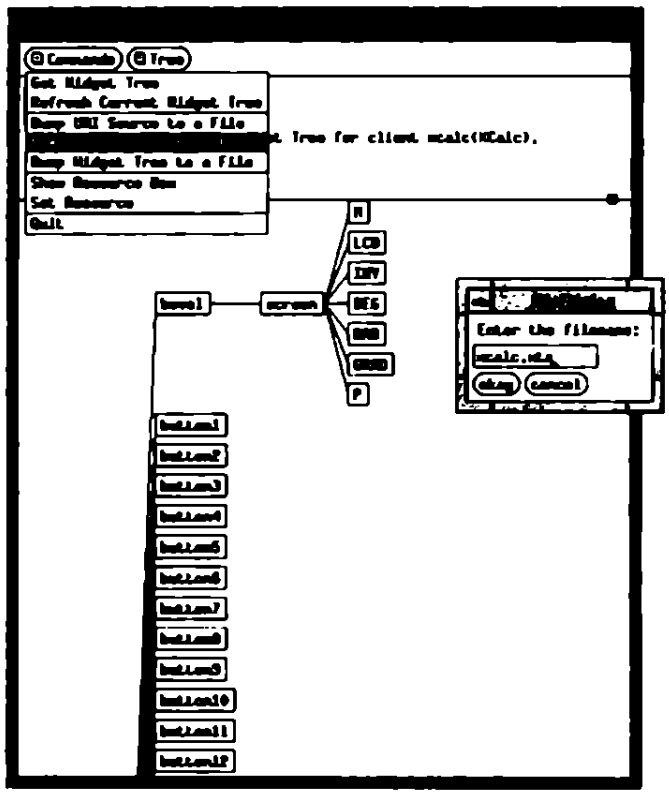

図 4.5: Analyzer

操作方法について簡単に説明する。

Commands メニューの項目の Get Widget Tree を選択すると，マウスポインタカーソルが十字型に変わる。調べたいGUIアプリケーションのウィンドウをクリックすると，そのアプリケーションがeditresプロトコルを実装している場合には，クライアントのウィジェットツリーが表示される。一度ウィジェットツリーが得られれば，他のメニュー項目を選択してクラス名やウィンドウIDのツリーを見たり，リソースの設定などを行なうことができる。これらは，全てeditres本来の機能である。

表示されたウィジェットツリーをXTSプログラムに変換するには，Commands メニューの "Dump XTSS source to a File" を選択する。ファイル名を問い合わせるダイアログボックスがポップアップされるので，ファイル名を入力すれば，そこへXTSプログラムが出力され

る。また，"Dump UAI source to a File" を選択すればUAI/Xプログラムが出力される。この部分が付加した機能であり，非常に簡単にXTSおよびUAI/Xプログラムを生成できるようになっている。ただし，Athena Widgetを使用していないGUIアプリケーションは，通常editresプロトコルが実装されていないので，全てのGUIアプリケーションには対応できないという問題がある。

図4.6にAnalyzerによってxcalcから生成されたXTSプログラムを示す。

```
Initialize xcalc XCalc
CreateManagedWidget ti formWidgetClass xcalc
CreateManagedWidget bevel formWidgetClass ti
CreateManagedWidget screen formWidgetClass bevel
CreateManagedWidget M labelWidgetClass screen
CreateManagedWidget LCD toggleWidgetClass screen
CreateManagedWidget INV labelWidgetClass screen
CreateManagedWidget DEG labelWidgetClass screen
CreateManagedWidget RAD labelWidgetClass screen
CreateManagedWidget GRAD labelWidgetClass screen
CreateManagedWidget P labelWidgetClass screen
CreateManagedWidget button1 commandWidgetClass ti
CreateManagedWidget button2 commandWidgetClass ti
CreateManagedWidget button3 commandWidgetClass ti
CreateManagedWidget button4 commandWidgetClass ti
CreateManagedWidget button5 commandWidgetClass ti
CreateManagedWidget button6 commandWidgetClass ti
CreateManagedWidget button7 commandWidgetClass ti
CreateManagedWidget button8 commandWidgetClass ti
    ...
CreateManagedWidget button40 commandWidgetClass ti
RealizeWidget xcalc
MainLoop
```

図4.6: Analyzerが生成したXTSプログラム

図4.7の左側にxcalcの概観，右側に図4.6のXTSプログラムにより表示された概観を示す。

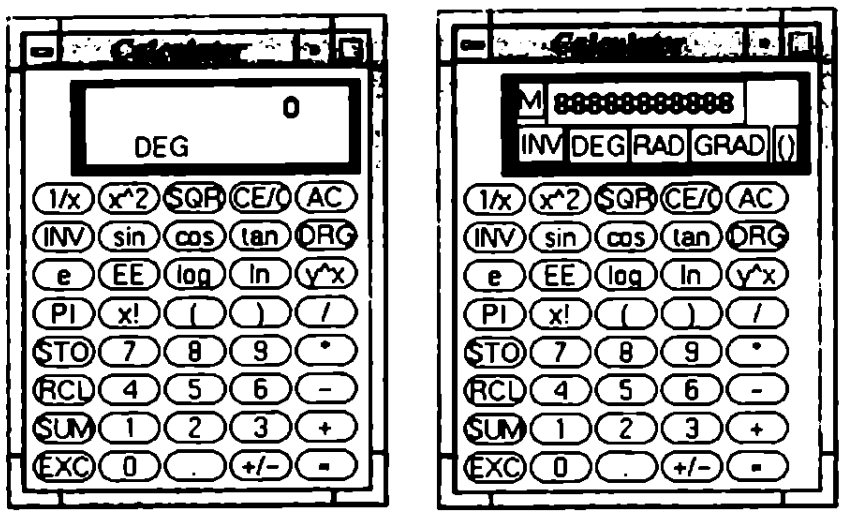

図 4.7: xcalc の比較

4.4.2 XTS プログラムの生成

editres は内部データ構造として，WNode 構造体を用いている。Get Widget Tree を実行すると WNode 構造体にウィジェット情報を格納するが，Analyzer はこの WNode に格納されている情報を基に XTS プログラムの生成を行なっている。

WNode 構造体は，ウィジェットの名前を示す name，ウィジェットのクラス名を示す class，ウィジェットの ID を示す id，ウィンドウの ID を示す window，親ウィジェットの WNode を指す parent，子ウィジェットの WNode の配列を指す children，その要素数を示す num_child という要素を持つ。

Analyzer は XTS および UAI/X プログラムを生成することができるが，ここでは特に XTS プログラムの生成に関して説明する。Analyzer による XTS プログラムの生成の手順は，以下の通りである。

1. ウィジェット階層のトップの WNode の name と class を取り出し，

   ```
   Initialize <name> <class>
   ```

   を出力する。

2. 子の WNode の class が composite クラスに属していれば CreatePopupShell，そうでなければ CreateManagedWidget 命令を生成する。

(a) classの先頭文字は大文字になっているので小文字に変換する。

(b) classからウィジェットであるかオブジェクトであるかを判断し，完全なウィジェットクラス名に変換する。すなわち，ウィジェットであれば<class>WidgetClass，オブジェクトであれば<class>ObjectClassなどに変換する。

以上のように変換した後，

```
CreateManagedWidget <name> <変換後のclass> <parent>
```

あるいは

```
CreatePopupShell <name> <変換後のclass> <parent>
```

と出力する。ただし，<parent>は親のWNodeの名前である。

3. 全ての子に対して同様の処理を再帰的に行なう。

4. 全ての子を生成したら，

```
RealizeWidget <トップのname>
MainLoop
```

を出力する。

4.5 評価

GUIの外観の生成については，部品の木構造から部品を選択し，外観を構成するウィンドウの親子関係の木構造を指定するという視覚的操作により，支援できることを示した。また，Athena Widgetにより生成されたGUIアプリケーションならば，その外観を生成するためのXTS簡易言語を自動的に生成でき，したがって，GUIのXTSSによる記述への移行を比較的簡単に行なえることも示した。

XTSS Builderの特徴として，以下のような点が挙げられる。

- XTSSによるTUIアプリケーションの統合
  XTSS Builderは，XTSSによる記述を視覚的に支援する補助的なシステムである。XTSS Builderでウィジェットの基本的な構成を定義し，XTSプログラムを生成し，その後XTSプログラムを修正し，XTCと組み合わせることによりTUIアプリケーションの統合を実現することができる。

- 視覚的なGUI記述環境

  XTSS Builderは，ツリーからツリーへの統一的な操作によって視覚的にGUIを構築する機能，GUIの構築中にいつでもGUIを表示して確認できる機能，設定可能なリソースの一覧を表示する機能，既存のアプリケーションのGUIを利用する機能等を提供しているため，視覚的操作で簡単にGUIの構築を行なうことが可能である。特に，既存のGUIアプリケーションをXTSSで再構築する場合には，Athena Widgetベースのものであれば，マウス操作だけで基本的なXTSプログラムを自動的に生成でき，迅速な再構築が可能である。

  このように，GUIの定義が迅速に行なえるようになるだけでなく，GUIを確認しながら少しずつ定義して行くことができるため，意図通りのGUIを誤り無く構築することが容易となる。ただし，初めに述べたように部品のイメージを直接張り付けながらGUIを定義するシステムと比較すると視覚化が不十分な面もある。

- GUI定義に要する知識の少なさ

  XTSS Builderを用いても，ウィジェットに関する知識，ウィジェットにおける各種リソースに関する知識は必要である。しかし，常にウィジェットの一覧が表示されており，各ウィジェットに関するリソースの一覧も指定により表示することができるため，利用者がこれらの情報を記憶しておく必要がない。単に一覧から選択するだけで済むため，記憶しておかなければならない知識の量がかなり少なくなる。

  また，Create(Managed)Widget, CreatePopupShellのようなウィジェット生成用の命令は自動的に生成されるため，これらのXTS命令については覚える必要がない。

Chapter 5

GUI分離型知的LISP言語学習システム

GUI分離型アプリケーションの適用例として，GUI分離型CAIシステムを提案する。CAIシステムをGUI分離型アプリケーションとして構築することにより，開発のコストを小さくし，汎用性・移植性の高いシステムとすることができる。このようなシステムを構築する際に，XTSSが非常に有効に利用できると期待される。本章では，XTSSを利用して，実際に既存の知的LISP言語学習システムからGUI分離型CAIシステムを構築することにより，XTSSがGUI分離型CAIシステムの構築に有効であることを示す。

5.1 CAIシステムとGUI

従来CAIシステムに関する研究は，学習コース，教材，学習者モデル，教授法等の教育内容に関連する部分に重点が置かれてきた。例えば，知的CAIシステム[Yamamoto87, Okamoto88, Okamoto90, Ohtsuki88, Toyoda88, Mizoguchi88]では，学習者の状態を学習者モデルで表現し，学習者モデルに基づきどのような指導を行なうかという教授法の研究が中心である。これに対して，学習者にユーザ・フレンドリな操作環境を提供するとともに，学習者の学習意欲を高めるために，CAIシステムにおけるユーザインタフェースに関する研究も活発に行なわれている。GUIはアイコン，メニュー，マウスなどを利用した視覚的な操作環境を提供するため，コンピュータに不慣れな学習者でも比較的簡単にCAIシステムを利用できるようになる。近年，コンピュータにはX WindowやMS-WindowsのようなGUIが標準的に装備されるようになってきたため，GUIを備えたCAIシステムが主流となってきている。

確かにGUIは学習者にとっては有効であるが，CAIシステム開発者にとっては次のような問題点がある。

1. CAIシステム開発の負担増

   GUIの記述は一般に非常に複雑で記述量も多くなりがちであるため，教育内容とは直接関係がないにも関わらず，その構築に多大の労力が必要となる。このため，教授法等のために十分な時間や労力を掛けることが困難になる。

2. 既存のCAIシステムの活用が困難

   従来のCAIシステムは，文字（テキスト）情報を主とするTUIが中心であった。TUIは，デバイスに対する依存性が低く，移植性も高いため，既存のシステムを活用することが比較的容易であったが，GUIは，例えばX WindowとMS-Windowsのように，それぞれの環境で記述方法がかなり異なっているため既存のシステムの活用は困難である。

5.2 GUI分離型CAIシステムの構想

これらの問題点を解決するための方法について考察する。

1.の問題に対しては，教育内容に直接関連する部分(CAI中核部と呼ぶことにする)とそれ以外の部分の開発を分離し，各部を別のグループで分担することが効果的である。しかし，モジュール分割のような通常の分担方法では，GUI部分が完成するまで動作試験ができないため，試行錯誤的に構築することが多い中核部の開発には適していない。これに対して，TUIに対応したCAI中核部を構築すれば，単独で動作できるため，試行錯誤的な構築も可能である。ただし，TUI部分の入出力の仕様はあらかじめ規定しておく必要がある。

2.の問題に対しては，CAIシステム(CAI中核部)をTUIアプリケーションとして構築すれば，その修正・応用は比較的容易となる。また，CAIシステムを異なるGUI環境に移植する際には，中核部は共通のものを用い，各環境に適したGUI部を追加すればよい。また，中核部を共通のコンピュータで動作させ，GUI部を別のGUIを有する端末上で実現することも可能となる。

このように，GUIに対応したCAIシステムを開発する際，以下のような部分から構成されるGUI分離型CAIシステムにより実現すれば問題点がかなり解決できることになる。

1. テキスト環境のみで独立に動作できるTUIに対応したCAI中核部
2. CAI中核部をGUI化するGUI部
3. CAI中核部とGUI部を連携させるために両者間の入出力データを制御する入出力制御部

XTSS は，2. を記述するための簡易言語を提供すると共に，1., 2., 3. の部分を結合するための機能を提供している。このため，XTSS を利用して GUI 分離型 CAI システムである知的 LISP 言語学習システムを構築する。

5.3 システムの概要

本システムは，図 5.1 に示すように，TUI アプリケーションであり CAI 中核部である知的 LISP 言語学習システム LISP-CAI，GUI 部であるウィンドウ生成部および XTS，LISP-CAI と GUI 部を連携させる入出力制御部，各部を接続する XTC から構成される。

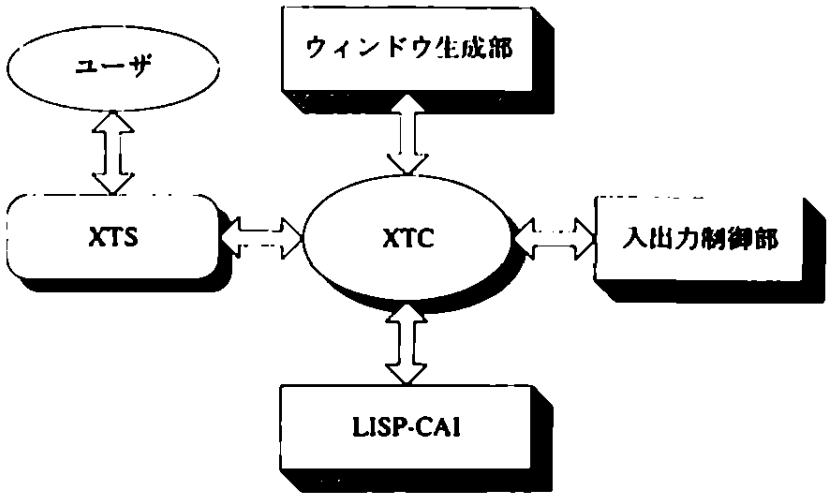

図 5.1: CAI システムの全体構成

LISP-CAI は，本システムの中核部となる TUI アプリケーションであり，テキスト環境のみで独立に動作でき，GUI 以外の CAI に関する全ての機能を提供する。LISP-CAI は XLISP で記述されている。ウィンドウ生成部はウィンドウの生成とイベント処理の指定を行なう部分であり，XTS 命令により記述されている。入出力制御部は，LISP-CAI とウィンドウ生成部（XTS）間のテキストデータの交換を管理する部分であり，テキストの処理が容易なシェル言語 perl で記述されている。

5.4 TUI 型知的 LISP 言語学習システム

初心者に演習を中心として LISP 言語教育を行なった際，あらかじめテキストに掲載されている順序で解答させると，途中で行き詰まり，先に進めなくなってしまう学生が見受けら

れる。しかし，行き詰まった問題を飛ばして少し傾向の違う他の問題を先に解くように指導し，後に飛ばした問題を再度解かせると解答できる場合がある。これは，苦手な問題を後回しにして他の問題を解いているうちに，苦手だった問題に関連する知識が増え，解答が可能になったことが原因であると考えられる。

本節では，このような問題を提示する順序に着目し，学習者が解きやすい問題から提示するという方法を採用したTUI型知的LISP言語学習システムLISP-CAIについて説明する。なお，本システムは，GUI分離型CAIシステムの有効性を示すためにCAI中核部の例として構築したものである。このため，知的CAIシステムとして最低限の機能である，学習者の理解状態に基づいて教授方法（ここでは問題を提示する順序）を動的に変更する機能は有しているが，学習者の誤り同定や助言機能については十分ではない面がある。

5.4.1 LISP-CAIの構成

LISP-CAIは，図5.2に示すように，知識ベース，学習者モデル，システムモジュール，ユーザモジュールの4つの部分から構成される。

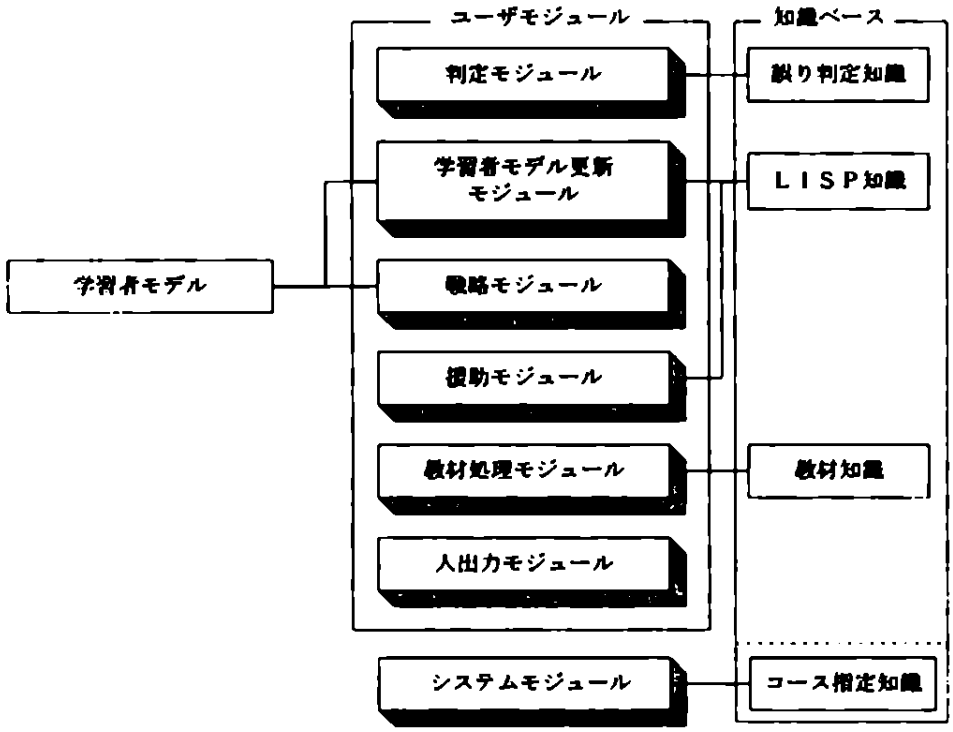

図5.2: LISP-CAIの構成

以下では各部分について簡単に説明する。

知識ベース LISP概念と関数データを木構造で保持するLISP知識，学習者に提示する解説および問題のデータベースである教材知識，教師が指定する学習コースを記述したコース指定知識，学習者の解答を模範解答とマッチングさせて正誤判定を行なうための誤り判定知識の4つの知識からなる。

学習者モデル 学習者の理解状態と学習履歴が保持されており，システムが学習者を指導するための基礎データとして利用する。

システムモジュール 教師が教材知識およびコース指定知識を参照してLISP-CAIの初期化を行なうためのモジュールである。

ユーザモジュール LISP-CAIのエンジン部分であり，学習者の解答の正誤判定を行なう判定モジュール，学習者モデルの更新を行なう学習者モデル更新モジュール，学習者モデルを参照し学習者に最適な次問選択を行なう戦略モジュールなどの6つのモジュールから構成される。

5.4.2 学習の流れ

LISP-CAIは，図5.3に示す学習の流れに従って実行される。

最初に，戦略モジュールが教師の指定した学習コースに従い，学習者モデルを参照して，学習者に最適な章・節・問題を決定する。教材処理モジュールは，新しい章・節に進んだ場合はその節の解説を，同じ節の場合は問題を教材知識から取り出し，学習者に提示する。その後，学習者からの入力待ち状態になる。

学習者が"ready"を入力すれば，解説が提示されている場合は問題の提示に進み，問題が提示されている場合は解答の判定に進む。援助機能コマンドが入力されれば，援助モジュールにより，章節の移動等の援助機能が実行され，学習者の入力待ち状態に戻る。それ以外の場合には，LISPプログラムと解釈し，LISPインタプリタにより入力を評価し，結果を出力した後，学習者の入力待ち状態に戻る。判定モジュールが，学習者の入力の履歴から解答を取り出し，正誤判定を行なう。最後に，正解あるいは不正解に応じて学習者モデル更新モジュールが学習者モデルを更新し，章・節・問題の選択に戻る。

なお，学習コースは後述するようにシステムモジュールによってコース指定知識から生成される。

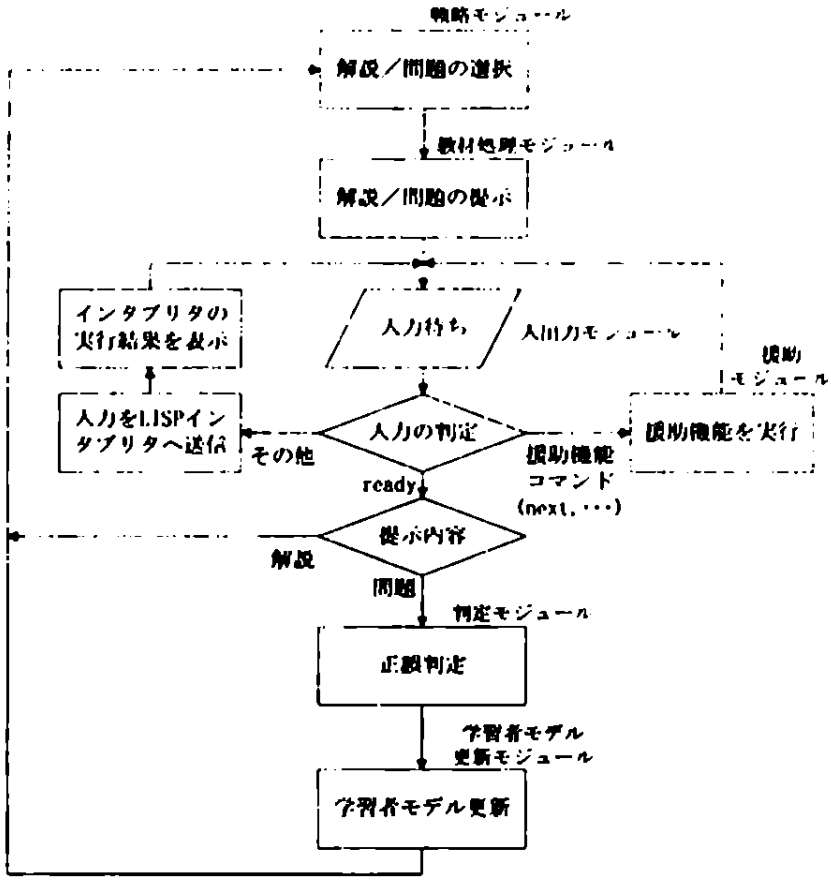

図 5.3: LISP-CAI の学習の流れ

5.4.3 知識ベース

知識ベースは，前述の4つの知識から構成されるLISP-CAIのデータベースである。

1. LISP知識

   LISP知識は，LISP概念木と関数データからなるLISP言語に関するデータベースである。LISP概念木は，図5.4のようにデータ型および文法に関する概念を木構造で記述したものである。なお，本システムではオーバレイモデルを採用しており，これらの各概念に学習者の理解度を付与することにより学習者モデルを表現している。

   関数データは関数の機能や用法に関する情報を各関数毎に記述したdictionaryおよびarg-functypeからなる。dictionaryは援助モジュールで利用される関数辞書のデータである。arg-functypeは下位の関数が呼び出される位置を示しており，学習者モデル更

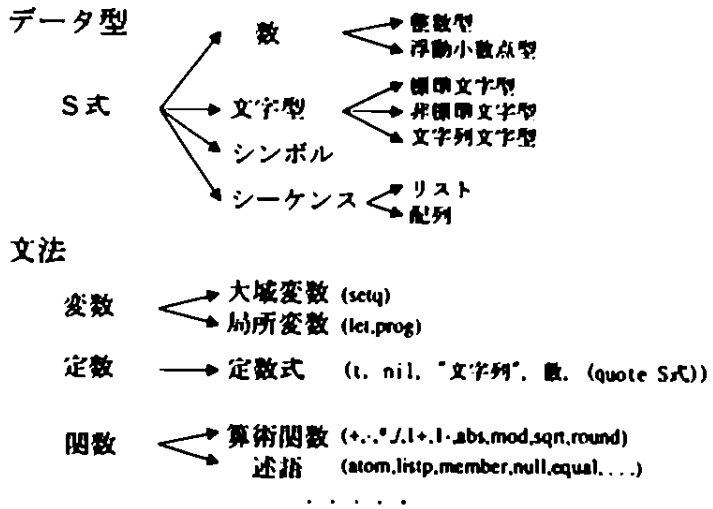

図 5.4: LISP 概念木

新モジュールで後述する寄与度を得るために使用される。

以下に，dictionary と arg-functype の記述例を挙げる。

dictionary の記述例 (cdr)

```
(cai-dictionary
   "この関数は一つの実引数を取り，その実引数はリストでなければ
   いけません。そのリストの尾を返します。すなわち，そのリスト
   から先頭要素を除いたリストを返します。
   例：(cdr '(a b c)) は (b c) を返します。")
```

arg-functype の記述例 (cond)

```
(cai-arg-functype ((func infinite) infinite))
```

ただし，func は関数が呼び出される引数位置であることを示し，infinite は一つ前の要素が任意回繰り返されることを意味する。

2. 教材知識

教材知識は，学習者に提示する解説と問題のデータベースで，解説知識と問題知識から構成される。以下に，解説知識と問題知識の記述形式および問題知識の記述例を挙げる。

解説知識の記述形式

```
exposition: 概要
statement:  学習内容の解説
```

問題知識の記述形式

```
exercise:  概要
statement: 問題文
answer:  模範解答のリスト
pattern: 問題パターン
data:    関数の引数のテストデータのリスト
hint:    問題に対するヒント
miscomment: 誤りの例
            誤りに対する助言
               :
```

問題知識の記述例

```
exercise: 算術関数の定義
statement: 2倍を返すLISP関数の...
answer: ((defun double (x)(* x 2)))
pattern: testdata
data: (5)
hint: 実数をn倍するLISP関数は...
miscomment: (defun double (x) (x * 2))
      関数を呼び出す場合リストの先頭に...
```

exposition と exercise にはその解説・問題の概要を示す単語が記述されている。pattern には問題のタイプが記述されている。タイプには，yes/no または数値等の単純な解答を求める one，LISP 関数を呼び出す解答を求める nodata，関数を定義させる testdata の 3 つがある。miscomment には，学習者が犯しやすい誤りの例とその誤りに対する助言の対を複数記述することができる。

3. コース指定知識

コース指定知識は，各教師が独自の演習順序を記述したもので，各章節において演習を行なう解説・問題の概要あるいはファイル名を指定する。以下に，記述形式および例を示す。

形式：(章番号) or (章番号 節番号) 表題
解説：概要あるいはファイル名 ...
問題：概要あるいはファイル名 ...

コース指定知識の記述例

(1) 関数
(1 1) 算術関数
解説: plus times ...
問題: plus1 plus2 times1 times2 ...

この知識は，学習者モデルにおいて学習者の選択した学習コースおよび学習履歴を保存するためにも利用される。

4. 誤り判定知識

学習者の解答は，模範解答とマッチングさせることにより正誤を判定する。誤り判定知識は，この際に学習者の解答と模範解答の双方を一定の標準的書式に従って書き換えるための知識(std-func)である。例えば，+, * は引数の順序が不定であるため，引数をソートして比較するという知識などがこの例である。以下に，記述例を挙げる。

std-funcの記述形式 (+)

```
(cai-std-func lisp-std-sort)
```

ただし，lisp-std-sortは引数をソートする関数である。

5.4.4 システムモジュール

システムモジュールは，図5.5に示すシステムの管理階層中のシステム管理者および教師が利用するモジュールである。

LISP-CAIでは，複数の教師が独自に作成した解説，例題，問題等の教材を全ての教師に共通の資源として教材知識に保持している。システム管理者は，この共有資源である教材知識の管理者であり，各教師が作成した教材を教材知識として登録・管理する。

教師はコース指定知識の管理者であり，教材知識から適切な解説・問題を選択し，章節を組み立てる。コース指定知識は，本モジュールにより実際に使用される学習コースとして初期化される。

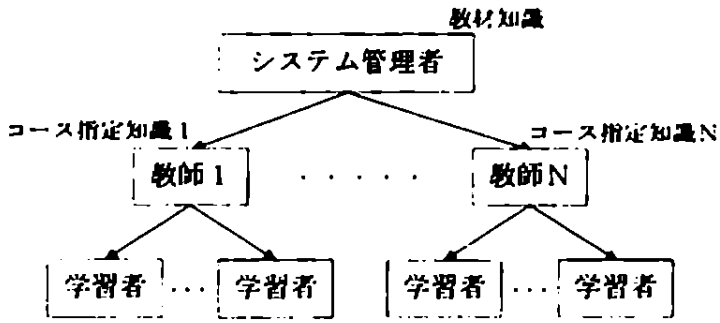

図 5.5: 管理階層

5.4.5 学習者モデル

学習者モデルは，学習者の各LISP概念に対する理解状況，学習コースおよび学習履歴により表現する。学習者はLISP概念木中の全ての概念を理解できるまで学習を行なうことになる。理解状況は，次の例のように各LISP概念にsuccess属性，fail属性，count属性を付与することによって記述している。

例：car概念　success属性　0.5
　　fail属性　0.5
　　count属性　0

success/fail属性は，それぞれ，その概念を含む問題に対する正解/不正解の度合いを表現している。count属性の値は学習者がその概念を含む問題を解答した回数である。問題に正解すればsuccess属性が，不正解ならばfail属性が増加するが，その操作については学習者モデル更新モジュールで述べる。

学習者のその概念に対する理解度は，success属性とfail属性を用いて，以下のように定義する。

$$\text{理解度} = \frac{success\text{属性}}{success\text{属性} + fail\text{属性}}$$

なお，理解度は，学習者にとって解きやすい問題を選択するために，戦略モジュールで使用される。

学習コースは，コース指定知識をリストで表現したものであり，参照済みの解説および問題には学習履歴情報としてマークが付与される。

5.4.6 ユーザモジュール

主要なモジュールについて説明する。

- 学習者モデル更新モジュール

  学習者の解答結果が正解か不正解かにより，success あるいは fail 属性の値を 1 増加させることにより学習者モデルの更新を行なうモジュールである。例えば関数定義の問題の場合には，関数データ arg-fuuctype を参照することにより問題に含まれている関数群を模範解答から自動的に抽出する。この関数群をその問題が含んでいる複数の概念（関数）であると判断し，関数の属性値の変化の総和が 1 になるように更新する。理解度が高いほど正解により大きく寄与し，理解度が低いほど不正解に大きく寄与すると考えられること，何度も使用した関数は安定状態にあり属性値の変化は小さくすべきであることを考慮し，次のように更新を行なう。

  [正解時]

$$\text{関数 } f\text{の寄与度} = \frac{\text{更新前の理解度}}{count\text{ 属性}}$$

$$\text{関数 } f\text{の } success\text{ 属性} = \text{更新前の } success\text{ 属性} + \frac{\text{関数 } f\text{の寄与度}}{\Sigma\text{関数の寄与度}}$$

  [不正解時]

$$\text{関数 } f\text{の寄与度} = \frac{1 - \text{更新前の理解度}}{count\text{ 属性}}$$

$$\text{関数 } f\text{の } fail\text{属性} = \text{更新前の } fail\text{属性} + \frac{\text{関数 } f\text{の寄与度}}{\Sigma\text{関数の寄与度}}$$

- 判定モジュール

  学習者の解答と模範解答とのマッチングを行ない正誤判定を行なうモジュールである。関数呼び出しおよび関数定義の問題では，正解が必ずしも一つとは限らないため，誤り判定知識により学習者の解答と模範解答の双方を一定の標準的書式に書き換えた後マッチングを行なう。なお，関数定義の問題では，仮引数部分にも任意性があるため，解答の仮引数部分を模範解答と合わせる処理も行なっている。学習者の解答が誤りであった場合，問題知識の miscomment に記述された誤りの例と比較し，一致する例が存在すれば，対応する助言を表示するという処理も行なう。

- 戦略モジュール

  学習者に出題する章・節・問題を決定するモジュールである。章・節は，学習コースに

従って決定する。問題は学習者モデルを参照し，学習者が最も解きやすいと予想される順に出題する。ここでは，学習者の理解度が低い関数をなるべく含まない問題が解きやすいと考え，各問題に使用されている関数の中の最低理解度が最も大きくなる問題を出題する。このように，学習者の誤りを直接明示的に診断し指摘するというよりはむしろ，各時点で学習者に最も理解しやすい問題を提示することにより，学習に対する敷居を低くすると共に，各学習者にとって少しずつ問題の難易度が上がって行くという効果が生まれ，学習がスムーズに進むことが期待できる。

- 援助モジュール

援助機能を提供するためのモジュールである。各援助機能と援助コマンドを表 5.1 に示す。

表 5.1: 援助機能・援助コマンド一覧

援助機能	援助コマンド
次問に進む	next
前問に戻る	prev
先頭に戻る	first
目次を見る	contents
章節の選択	move
再表示	show
編集	edit
ヒント	hint
関数用法	dict
システムの使用法	help
終了	bye

「次問に進む」と「前問に戻る」は，学習者に問題の選択権を与えるための機能，「先頭に戻る」は未解答の最初の問題に戻るための機能である。「目次を見る」「章節の選択」は，学習者が学習コースに沿わずに章節を移動するための機能である。「再表示」は解説あるいは問題を表示し直す。「編集」は，ファイル上での解答作成を可能にするためにテキストエディタを開き，エディタを終了すると作成されたファイルを自動的に学習者の入力として読み込む。「ヒント」は問題知識の hint に記述されたヒントを表示する。「関数用法」は LISP 知識の dictionary に記述された関数の使用法を表示する。「終了」は，LISP-CAI の終了処理を行なう。

5.5 GUI型知的LISP言語学習システム

LISP-CAIを例としてGUI分離型CAIシステムを構築する際に，以下のような設計を行なった。

1. マルチウィンドウ

   LISP-CAIでは，一度解説を読んだ後は，問題を解く時に解説を参照することができない。このため，解説表示ウィンドウと問題表示ウィンドウをマルチウインドウで常に表示する。

2. マウス

   CAIによる学習の場合，複雑なコマンド入力やキー操作が学習者の学習の妨げになるケースが非常に多いことが指摘されている。このため本システムでは，可能な限り全ての操作をマウスで行なえるようにする。

3. メニュー

   LISP-CAIでは，章節の選択の際に，章節の一覧が表示され，選択する章節の番号を入力しなければならなかった。このため，章節の一覧メニューを提供し，マウスにより選択できるようにする。

5.5.1 システムの概要

上記の各項目を満足するために図5.6のようなウィンドウ構成のシステムとした。

1. トップメニュー

   システムが最初に起動されると，LISP-CAIから出力される選択肢に基づきトップメニューを構成し表示する。現在，選択肢は「前回止めた場所から」「始める章，節を選択する」「新しく始める」であるが，LISP-CAI側の選択肢が変更されても自動的にトップメニューが変更されるように設計されている。トップメニューおよびシステム全体の外観を図5.7に示す。

2. 解答ウィンドウ

   解答完了および終了などの援助機能用のボタン群，判定結果を表示する判定ウィンドウ，実行結果などの各種メッセージを表示するメッセージウィンドウ，解答を入力するための解答入力ウィンドウから構成される。図5.8にウィンドウの外観を示す。

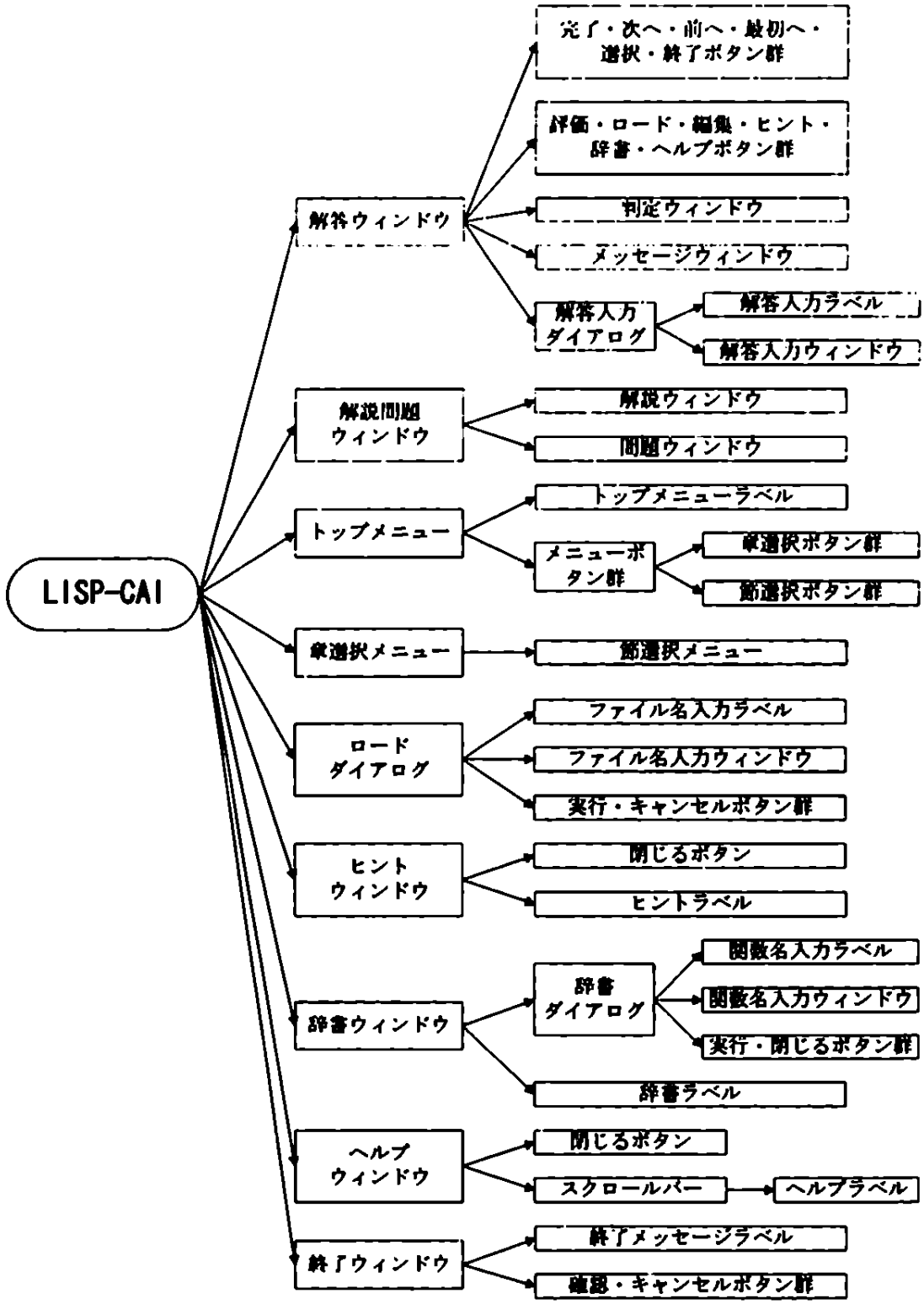

図 5.6: ウィンドウ全構成

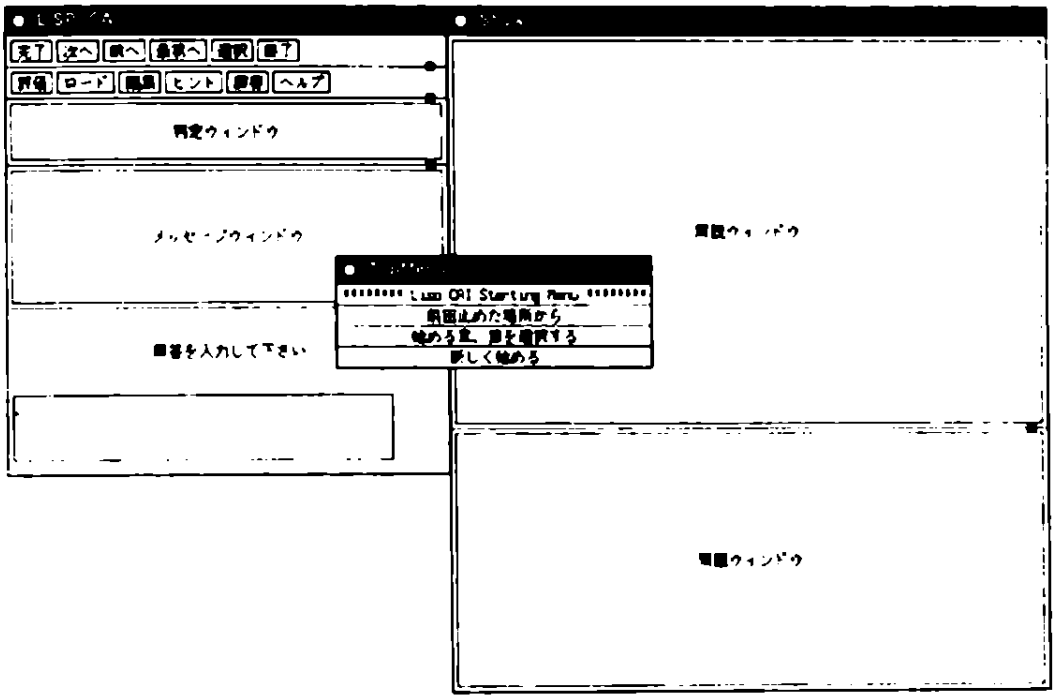

図 5.7: ウィンドウ全体の外観

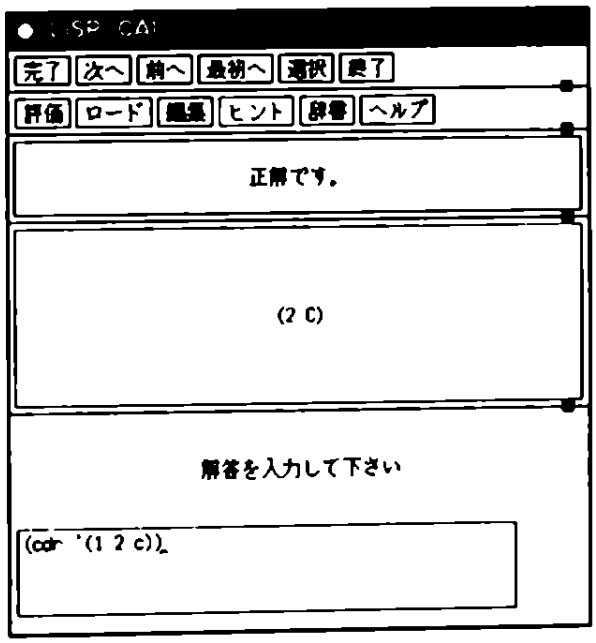

図 5.8: 解答ウィンドウの外観

完了ボタンは ready，次へは next，前へは prev，最初へは first，選択は move，終了は bye の各援助コマンドに対応している。選択ボタンが押されると章節選択メニューが出現する。評価ボタンは解答入力ウィンドウに入力されたデータを LISP 式として実行しその結果をメッセージウィンドウに表示する。これにより補助関数のテストや変数値の確認など，LISP プログラムをデバッグする際にインタプリタ上で行なう各種操作を実現できる。ロードは指定されたファイルをロードする。編集はエディタを起動し，エディタが終了すると編集したファイルの内容をロードする。ヒントは hint，辞書は dict，ヘルプは help の各援助コマンドに対応している。

3. 解説問題ウィンドウ

解説および問題を表示するウィンドウであり，解説と問題が同時に表示されるため学習者は解説を確認しつつ学習を進めることができる。図 5.9 にウィンドウの外観を示す。

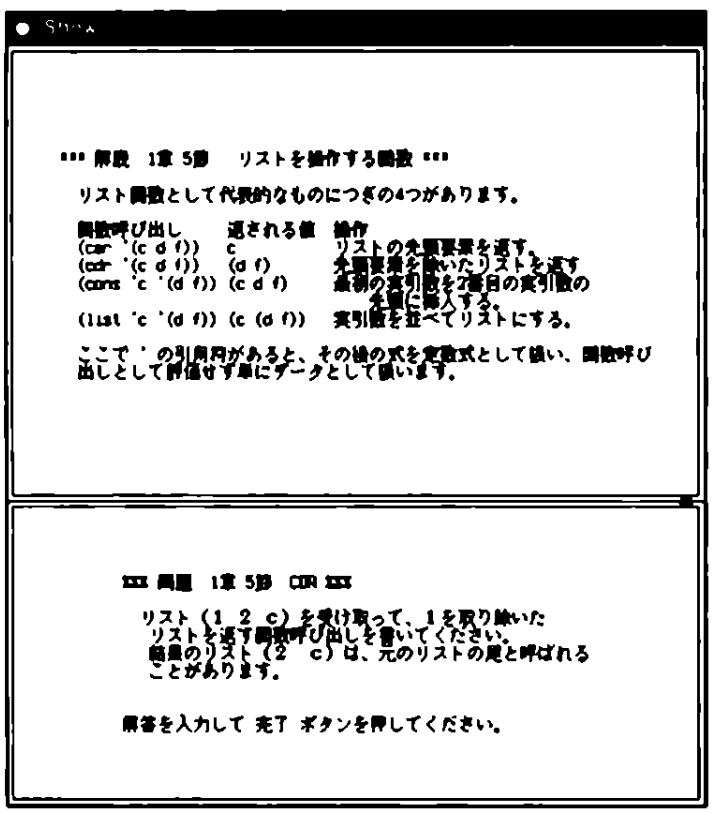

図 5.9: 解説問題ウィンドウの外観

4. 章節選択メニュー

選択ボタンを押すと章選択メニューが表示される。図5.10に示すように，選択したい章のタイトル上でボタンを押すと，節選択メニューが出現する。選択したい節のタイトルでボタンを離せば選択された章節の解説あるいは問題が表示される。これらのメニューはcontentsコマンドの出力から構成している。

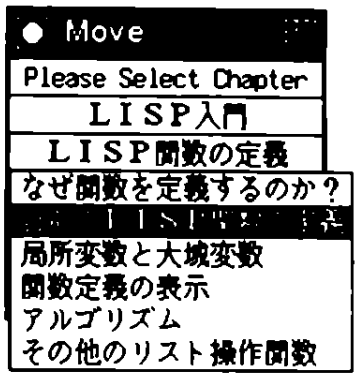

図5.10: 章・節選択メニューの外観

5. ロードダイアログ

ロードダイアログは，ファイル名入力ウィンドウにファイル名を入力し，実行ボタンを押すとそのファイルがロードされ，キャンセルボタンを押すとポップダウンされる。図5.11に外観を示す。

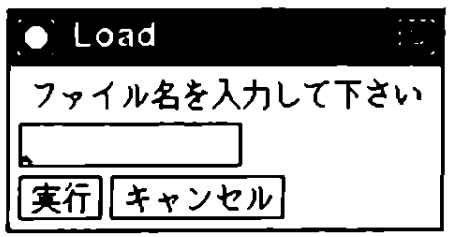

図5.11: ロードダイアログの外観

6. ヒントウィンドウ

hintコマンドにより得られる，その問題に対するヒントを表示し，「閉じる」ボタンを押すとポップダウンする。図5.12に外観を示す。

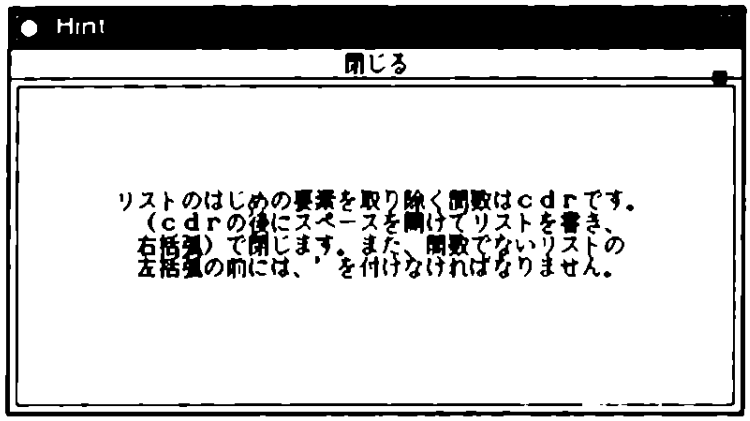

図 5.12: ヒントウィンドウの外観

7. 辞書ウィンドウ

関数名入力ウィンドウに関数名を入力し実行ボタンを押すと dict コマンドにより得られるその関数に関する情報を表示する。「閉じる」ボタンを押すとポップダウンする。図 5.13 に外観を示す。

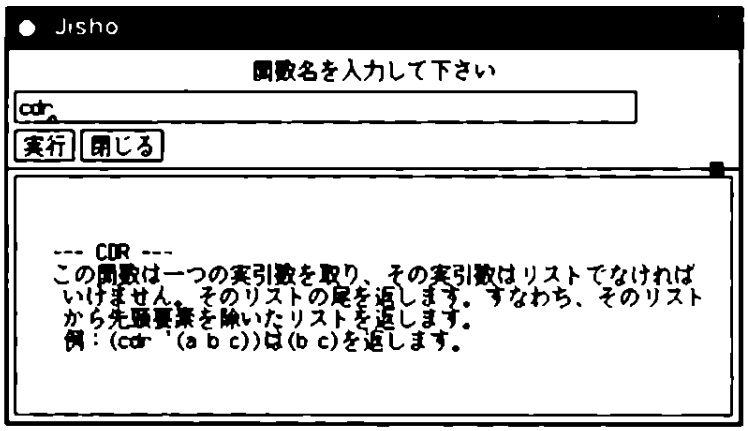

図 5.13: 辞書ウィンドウの外観

8. 終了ウィンドウ

終了確認のためのウィンドウである。

5.5.2 ウィンドウ生成部

ウィンドウ生成部は，CAI中核部と連携するGUI部で，XTS簡易言語で記述され，前に述べた各ウィンドウを生成するとともに，ウィンドウ上で起こるイベント，ここではボタンイベントを入出力制御部に通知するための処理を行なう。

各ボタンに関するイベント処理を簡単に説明する。各ボタンには主として，どのボタンが押されたかを示すテキストを標準出力に出力する命令がコールバック関数として登録されており，XTCを介して入出力制御部に伝送される。例えば，完了ボタンを押すと"ready"という文字列が，ロードボタンを押すとロードダイアログがポップアップし，ファイル名を入力して実行ボタンを押すと"load ファイル名"という文字列が入出力制御部に送られる。表5.2に，ボタンとその動作の対応を示す。

表 5.2: ボタンとその動作

ボタン	動作
完了ボタン	"ready" の出力
次へボタン	"next" の出力
前へボタン	"prev" の出力
最初へボタン	"first" の出力
選択ボタン	"move" の出力
終了ボタン	終了ウィンドウのポップアップ
確認	"bye" の出力と終了ウィンドウのポップダウン
キャンセル	終了ウィンドウのポップダウン
評価ボタン	"eval" の出力
ロードボタン	ロードダイアログのポップアップ
実行	"load ファイル名" の出力とロードダイアログのポップダウン
キャンセル	ロードダイアログのポップダウン
編集ボタン	"edit" の出力
ヒントボタン	ヒントウィンドウのポップアップと"hint" の出力
閉じる	ヒントウィンドウのポップダウン
辞書ボタン	辞書ウィンドウのポップアップ
実行	"dict 関数名" の出力
閉じる	辞書ウィンドウのポップダウン
ヘルプボタン	ヘルプウィンドウのポップアップと"help" の出力
閉じる	ヘルプウィンドウのポップダウン

5.5.3 入出力制御部

入出力制御部は，LISP-CAIおよびウィンドウ生成部（XTS）から送られてくるデータを読み込み，解析し，解析結果に基づいてLISP-CAIあるいはXTSに適切なデータを送り返す。図5.14に入出力制御部の処理の流れを示す。

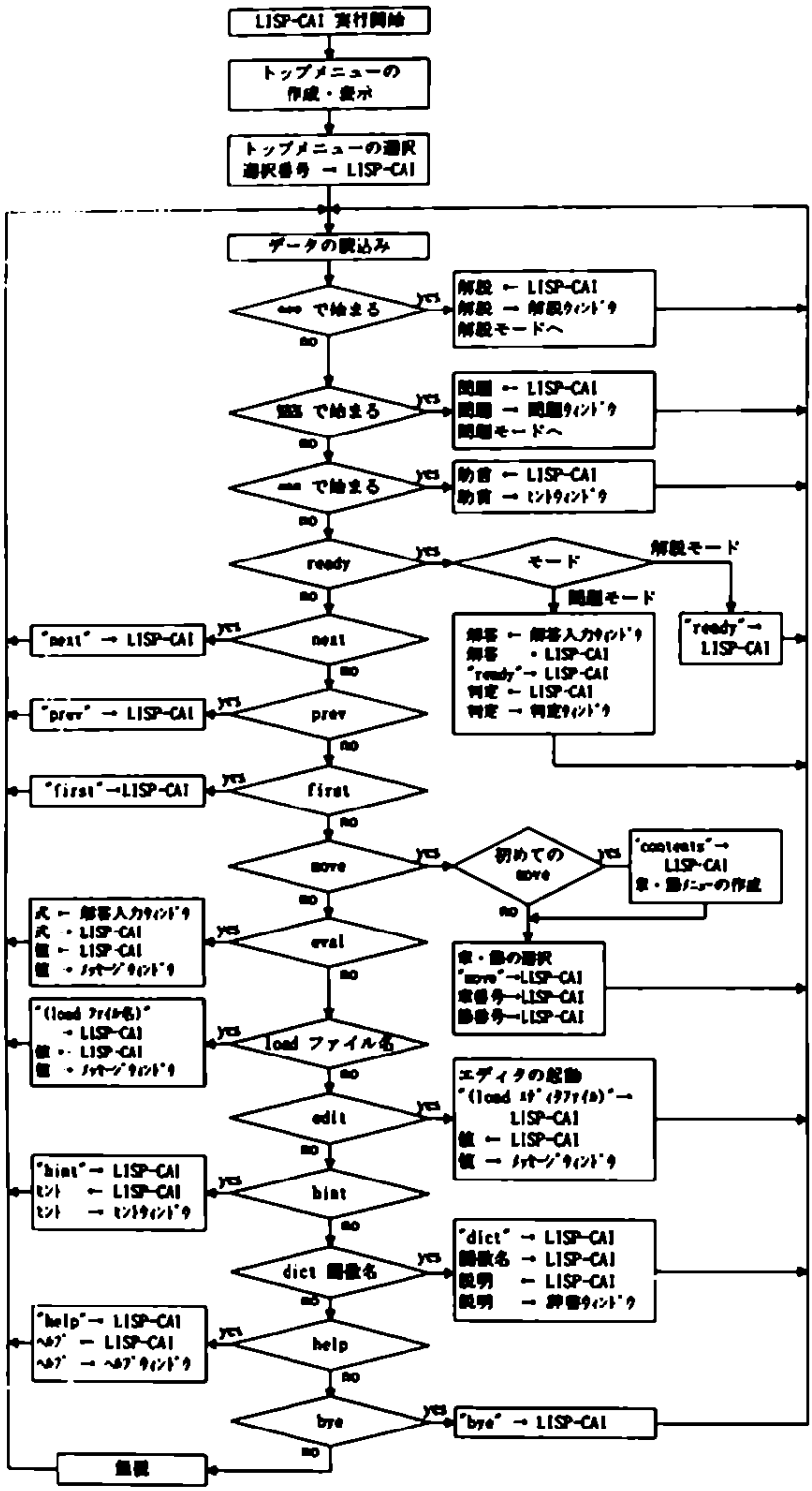

図 5.14: 入出力制御部の処理の流れ

処理の流れを以下に簡単に説明する。

1. LISP-CAIに実行開始を通知する。
2. LISP-CAI側から送られてきたデータに基づきトップメニューを構成し表示する。
3. 学習者の選択したメニューに基づき選択された番号をLISP-CAIに通知する。
4. LISP-CAIあるいはXTSからXTCを介して送られてきたデータを読み込む。
5. 解説の処理
   データが *** で始まっていれば，解説の表示なので，解説を終わりまで読み込み，それを解説ウィンドウに表示し，解説モードに移行する。
6. 問題の処理
   データが %%% で始まっていれば，問題の表示なので，問題を終わりまで読み込み，それを問題ウィンドウに表示し，問題モードに移行する。
7. 助言の処理
   データが === で始まっていれば，助言の表示なので，助言を終わりまで読み込み，それをヒントウィンドウに表示する。
8. readyの処理
   解説モードならばreadyをLISP-CAIに送る。問題モードならば，解答入力ウィンドウのデータとreadyをLISP-CAIに送り，LISP-CAIからの出力を判定ウィンドウに表示する。
9. next, prev, firstの処理
   データをそのままLISP-CAIへ送る。
10. moveの処理
    章節選択メニューをポップアップして学習者の選択を待ち，選択された章節の番号をLISP-CAIに送る。はじめての処理の場合にはcontentsコマンドをLISP-CAIに送り，送り返された章節の一覧に基づいて章・節メニューを作成する。
11. evalの処理
    解答入力ウィンドウのデータをLISP-CAIに送り，返された値をメッセージウィンドウに表示する，

12. 「load ファイル名」の処理
    (load "ファイル名") を LISP-CAI に送り，返されたメッセージをメッセージウィンドウに表示する。

13. edit の処理
    edit.l というファイルを対象にエディタを起動し，編集の終わりを待つ。(load "edit.l") を LISP-CAI に送り，返されたメッセージをメッセージウィンドウに表示する。

14. hint の処理
    hint を LISP-CAI に送り，返されたメッセージをヒントウィンドウに表示する。

15. 「dict 関数名」の処理
    LISP-CAI に dict および関数名を送り，返されたメッセージを辞書ウィンドウに表示する。

16. help の処理
    help を LISP-CAI に送り，返されたメッセージをヘルプウィンドウに表示する。

17. bye の処理
    bye を LISP-CAI に送り，システムを終了する。

5.6 評価

GUI分離型CAIシステムの構築を提案し，TUIアプリケーションである LISP-CAI をCAI中核部とし，GUI部（ウィンドウ生成部）をXTSで，入出力制御部を perl で記述することによりGUI対応のCAIシステムを構築した。この結果，

1. GUI部について，以下のように当初の設計を全て満足できた。

   (a) 解説と問題の同時表示を実現した。

   (b) 解答入力および評価したいLISP式の入力，ロードしたいファイル名の入力，辞書引きをしたい関数名の入力以外の操作は全てマウスによる指定が可能となった。

   (c) 章節選択のメニュー化を実現した。

2. LISP-CAI システムを修正することなくGUI化を実現したため，元の LISP-CAI システムは依然として単独でテキスト環境で利用することができる。また，LISP-CAI シ

ステムはテキスト入出力に基づいているため比較的修正が容易であり，試行錯誤的な改良にも適している。

3. ウィンドウ生成部および入出力制御部の設計・製作はLISP-CAIシステムの仕様に基づいて独立に行なうことができた。また，LISP-CAIの教授法等が変更されてもテキストレベルの入出力の仕様が変更されない限り，GUI関連の部分は無修正で利用できる。

4. XTSとXTCを用いることによって，ウィンドウ生成部と入出力制御部はC言語などのプログラミング言語ではなく，簡易言語XTS (130行) とシェル言語 (420行) により記述できたため，短期間に開発でき，少ない記述量で実現できた。

したがって，XTSSはGUI分離型CAIシステムの構築に有効であると考えられる。

Chapter 6

結論

本論文では，GUIアプリケーションを構築するために，基本的な機能を提供するTUIアプリケーションを作成し，それらを統合するGUIプログラムを定義するという方法を提案した。さらに，このようなGUI分離型アプリケーションを構築するために，TUIアプリケーションのGUI環境下への統合を支援する統合化支援システムとして，宣言的に記述を行なうUAI/X，手続き的に記述を行なうXTSS，視覚的に構築を援助するXTSS Builderを提案し，これらのシステムがGUI分離型アプリケーションの構築に有効であることを示した。特にXTSSについては，TUIアプリケーションである知的LISP言語学習システムをGUI分離型CAIシステム，すなわちGUIアプリケーションに統合する例を示し，統合化支援システムがかなり実用的に利用できることも示した。以下では，各システムについて残された問題点や今後の課題について述べる。

UAI/X

UAI/Xの今後の課題として，以下の項目が考えられる。

- 実行速度を向上させるために簡易言語をコンパイルする機能の付与
- ユーザが視覚的な指定によって簡易言語を生成できる援助ツールの提供
- エディタやデバッガのようにより高度な入出力の同期が必要となる複雑な連続型複合動作アプリケーションの統合
- C言語プログラムの一部として組み込みが可能となるようなウィンドウの属性をファイルとみなす視点の導入

これらの課題を解決することでより多種多様なアプリケーションがより容易に統合可能となることが期待できる。

XTSS

XTSSには以下のような問題点がある。

- イベントハンドラを提供していない
- コールバック関数におけるクライアントデータの記述方法があまり容易でない
- グラフィックコンテクスト，カラー，その他Xのレベルの機能をより充実させる必要がある

今後の課題としては以下の項目が考えられる。

- 常駐するシェルの実現
  サーバ側でシェルを常駐し，クライアント側からあらかじめ命令を送っておけば，後は自律的に実行を続けることができる。これはサーバが他のホストで実行されている場合に有効であると思われる。
- グループウェアへの対応
  1つのサーバで複数のクライアントに対応できるようサーバを改良する。このサーバを用いて複数人がウィンドウ環境を共有し，協調作業ができるような機能を付与する。

XTSS Builder

今後の課題として以下のようなものが挙げられる。

- XTSプログラムの解析の充実
  ウィジェット生成用命令にしか対応できていない。XTSプログラムを直接記述した場合にも対応できるようにこれ以外の命令についても解析し，修正を援助し，そして出力できる機能を完備する必要がある。
- 誤り検出および訂正機能
  ウィジェットの親子関係を定義するときに子を持てないクラスのウィジェットに子を繋いだり，あるいは，リソース値として不適切な値を指定してしまうことがある。現在

のところこのようなエラーの判定は全てXTSSに任せているが，XTSS Builder自体で適宜エラーメッセージを出力したり，エラーを自動的に訂正する機能が必要である

GUI分離型CAIシステム

以下のような問題点が残されている。

- 処理速度

  シェル言語はインタプリタで実行されるため，実行速度がやや遅い。このため，計算機の負荷が高い時などにボタン操作を高速に連続して行なうとプロセス間通信のためのバッファが溢れるなどして問題を起こすことがある。これを解決するためには，他の処理中にボタン操作を無視するような機能が必要であるが，XTS ではこのような機能は提供されていない。

- デバッグ

  ウィンドウ制御部は各ウィンドウの機能とコールバック関数で出力する文字列とを設計することが中心のため，デバッグは容易である。しかし入出力制御部は，ウィンドウ制御部とCAI中核部からの両方のデータを対象としているため，デバッグはそれ程容易ではない。

今後の課題としては，以下のような項目が考えられる。

- CAI中核部とGUI部の完全分離

  現在CAI中核部とGUI部は同じ計算機上で実行されているが，CAI中核部をワークステーション上で実現し，GUI部をネットワーク経由でワークステーションに接続されたパーソナルコンピュータ上で実現することにより完全に分離し，負荷分散を促進する。

- クラスルーム型CAIシステムへの応用

  複数人でグループ学習を行なうテキスト型のCAIシステムを構築し，それをGUI分離型の概念によりGUI化するための研究を行なう。

参考文献

[Aguiu92] 安上.院猛, 永江孝規: Xアプリケーション・プログラミング2, 新紀元社(1992)

[Almy92] Almy,T.: XLISP-PLUS: Another Object-oriented Lisp (1992)

[Hoff96] Hoff, A.v., Shaio,S., Starbuck,O.: HOOKED on JAVA, Addison-Wesley Publishing Company (1996)

[Ida91] 井田昌之翻訳監修: COMMON LISP 第2版, 共立出版(1991)

[Isoda87] Isoda, Shimomura, Ono: VIPS: A Visual Debugger, IEEE Software, Vol. 4, No. 3, pp. 8-19 (1987)

[Kabutogi90] 兜木昭男, 木下凌一他: X-Window OSF/Motifプログラミング, 日刊工業新聞社(1990)

[Kuno88] 久野靖, 角田博保: 流れて行かないUNIX環境, 情報処理学会論文誌, Vol. 29, No. 9, pp. 854-861 (1988)

[Kuno89] 久野靖: 新しいプログラマ・インタフェースの利用, 情報処理, Vol. 30, No. 4, pp. 396-405 (1989)

[Lemay95] Lemay, L.: Teach Yourself Web Publishing with HTML in a Week, Sams Publishing (1995)

[Mayer90] Mayer,N.P.: The WINTERP Widget INTERPreter An Application Prototyping and Extension Environment for OSF/Motif, Proceedings X into The Feature, The European X Users Group Autumn Conference 1990, pp. 35-55 (1990)

[McCormack88] McCormack,J. et al.: X Toolkit Intrinsics - C Language Interface, X Window System, X Version 11, Release 4, MIT (1988)

[McCormack91] McCormack,J. et al.: X Toolkit Intrinsics - C Language Interface, X Window System, X Version 11, Release 5, MIT (1991)

[Mizoguchi88] 溝口理一郎, 角所収: 知的CAIにおける学習者モデル, 情報処理, Vol. 29, No. 11, pp. 1275 1282 (1988)

[Nakade95] 中出雅子, 阪井節子, 高濱徹行: グラフィカルユーザインタフェース構築支援システム XTSS の改良, 福井大学情報処理センター NETWORK, Vol. 9, No. 3, pp. 17 26 (1995)

[Neumann93] Neumann, G., Nusser, S.: Wafe - An X Toolkit Based Frontend for Application Programs in Various Programming Languages, Proceedings of the 1993 Winter USENIX Conference (1993)

[Ohtsuki88] 大槻説乎, 山本米雄: 知的CAIのパラダイムと実現環境, 情報処理, Vol. 29, No. 11, pp. 1255 1265 (1988)

[Okamoto88] 岡本敏雄: 知的CAI, 電子情報通信学会誌, Vol. 71, No. 4, pp. 384-390 (1988)

[Okamoto90] 岡本敏雄, 溝口理一郎監訳: 知的CAIシステム, オーム社(1990)

[Olsen92] Olsen,D.R.: User Interface Management Systems: Models and Algorithms, Morgan Kaufmann Publishers, Inc. (1992)

[OSF90] Open Software Foundation: OSF/Motif Programmer's Guide, Revision 1.1, Open Software Foundation, Inc. (1990)

[Ousterhout90] Ousterhout, J.K.: Tcl: An Embeddable Command Language, Proceedings of the 1990 Winter USENIX Conference (1990)

[Ousterhout91] Ousterhout, J.K.: An X11 Toolkit Based on the Tcl Language, Proceedings of the 1991 Winter USENIX Conference (1991)

[Peterson89] Peterson,C.D.: Athena Widget Set - C Language Interface, X Window System, X Version 11, Release 4, MIT (1989)

[Peterson91] Peterson,C.D.: Athena Widget Set - C Language Interface, X Window System, X Version 11, Release 5, MIT (1991)

[Petzold89] Petzold,C.: プログラミング Windows Vol. 1-2, アスキー出版(1989)

[Rekimoto90] 暦本純一, 垂水浩幸他: エディタを部品としたユーザインタフェース構築基盤: 前, 情報処理, Vol. 31, No. 5, pp. 602-611 (1990)

[Sakai94] 阪井節子, 高濱徹行: グラフィカル・ユーザインタフェース分離型 LISP-CAI システムの構築, 教育工学関連学協会連合第4回全国大会論文集, pp. 595-596 (1994)

[Scheifler89] Scheifler,R.W., Oren,L., et al.: CLX Programmer's Reference, Texas Instruments Inc. (1989)

[Sugimoto89] 杉本直樹, 武田博隆他: X Toolkit 上でのアプリケーションプログラム作成支援ツール, 情報処理学会第39回全国大会, pp. 1151-1152 (1989)

[Takahama90] 高濱徹行, 中谷洋一, 小倉久和, 中村正郎: 応用プログラム群を統合するウィンドウ型ユーザインタフェースシステム, 平成2年度電気関係学会北陸支部連合大会, p. 119 (1990)

[Takahama91] 高濱徹行, 中谷洋一, 小倉久和, 中村正郎: 応用プログラム群を統合するウィンドウ型ユーザインタフェースシステム: UAI/X, 情報処理学会第42回全国大会 (5), pp. 33-34 (1991)

[Takahama92] 高濱徹行, 中谷洋一, 小倉久和, 中村正郎: 応用プログラム群を統合するウィンドウ型ユーザインタフェースシステム: UAI/X, 電子情報通信学会論文誌, Vol. J75-D-I, No. 7, pp. 479-487 (1992)

[Takahama93a] 高濱徹行, 八木正和, 小倉久和, 中村正郎: ウィンドウ型ユーザインタフェース記述システム: XTSS, 平成5年度電気関係学会北陸支部連合大会, p. 269 (1993)

[Takahama93b] 高濱徹行: X ツールキットサービスプロトコルによる X ウィンドウプログラミング, 福井大学情報処理センター NETWORK, Vol. 7, No. 1, pp. 19-44 (1993)

[Takahama93c] 高濱徹行, 八木正和, 小倉久和, 中村正郎: グラフィカルユーザインタフェース構築支援ツールの開発, 福井大学工学部研究報告, Vol. 41, No. 2, pp. 199-208 (1993)

[Takahama94] 高濱徹行, 小倉久和, 中村正郎: テキスト型アプリケーション群を統合するグラフィカルユーザインタフェースシステム: XTSS, 電子情報通信学会論文誌, Vol. J77-D-I, No. 7, pp. 493-502 (1994)

[Takahama97] 高濱徹行, 牧野とみ, 阪井節子: CAI システムにおけるグラフィカル・ユーザインタフェースの分離手法, 教育システム情報学会誌, Vol.13, No.4 (1997) 掲載予定

[Takahashi89] 高橋健司, 下村隆夫他: 既存デバッグツールを利用した高速なビジュアルデバッグ方式, 情報処理学会第38回全国大会, pp. 1235-1236 (1989)

[Toyoda88] 豊田順一, 中村祐一: 知的CAIにおける知識表現と教授法, 情報処理, Vol. 29, No. 11, pp. 1266-1274 (1988)

[Wall90] Wall,L.,Schwartz,R.L.: Programming perl, O'Reilly and Associates, Inc. (1990)

[Yamamoto87] 山本米雄, 岡本敏雄監訳: 人工知能と知的CAIシステム, 講談社(1987)

[Yokoyama89] 横山孝典, 谷正之他: 対話の階層モデルに基づくユーザインタフェース管理システム, 情報処理学会論文誌, Vol. 30, No. 4, pp. 507-517 (1989)

[Young89] Young,D.A.: X Window Systems: Programming and Applications with Xt, Prentice-Hall (1989)

Appendix A

XTS 命令一覧

各 XTS 命令について簡単に解説する。

特殊命令

- CALL コマンド 引数 ...

  コマンドを実行する。コマンドからの出力がメッセージとして返される。

  (例) 現在のディレクトリが /home/taro の時，ディレクトリ名を返す。

  ```
  CALL pwd
  OK /home/taro
  ```

- CD ディレクトリ

  現在のディレクトリを変更する。変更に失敗した時はエラーとなり，成功した時は現在のディレクトリを返す。

  (例) 現在のディレクトリを /home/taro に変更する。

  ```
  CD /home/taro
  OK /home/taro
  ```

- DATA 引数 ...

  XTS には $0, $1, ... のように位置で表現する位置変数と名前を持つ名前付き変数が存在する。DATA 命令は $0 を 'OK' とし，$1 以降の位置変数に各引数を代入する。

  (例) Hello を第一位置変数，World を第二位置変数とする。

  ```
  DATA Hello World
  OK                              $0: OK, $1: Hello, $2: World
  ```

- ECHO 引数 ...

  引数をそのままメッセージとして返す。結果コードは返さない。

  (例) Hello World をメッセージとして返す。

```
ECHO Hello World
Hello World
```

- ELSE XTS 命令

  直前の IF 命令 (後述) が偽となった時に，XTS 命令を実行する．

  (例) 偽の場合に ECHO 命令を実行する。

```
ELSE ECHO Hello World
Hello World                          偽の場合
OK                                   真の場合
```

- ENTRY タイプ 名前

  指定したタイプの名前のハッシュ番号を返す。デバッグ用の命令である。

- EVAL 引数 ...

  引数を XTS 命令として実行し，その結果コードとメッセージを位置変数に代入する。

  (例) Hello World を返し，それを位置変数にする。

```
EVAL ECHO Hello World
Hello World                          $0: Hello, $1: World
<ECHO 命令の結果である "Hello World" を位置変数に代入した>
```

- EXEC 引数 ... 引数

  各引数をそれぞれ XTS 命令として左から順に実行する。

  (例) Hello を出力し，World を出力する。

```
EXEC "ECHO Hello" "ECHO World"
Hello
World
```

- EXIT [{+|−}{echo|fatal|error|ok|all}] ...

  XTS応答の結果コードによってXTSの実行を終了するための終了モードを指定する。+ は終了することを，− は終了しないことを指定し，echo, fatal, error, ok は結果コードがそれぞれに対応するメッセージのみに終了操作を行ない，all は全てのメッセージに終了操作を行なうことを指定する。初期は EXIT +fatal であり，Fatal メッセージの際に XTS の実行を終了する。なお，引数がない時は現在の終了モードの状態を出力する。

  (例) 結果コード が Fatal の時でも XTS の実行を続ける。

```
EXIT -fatal
OK
```

- FORK コマンド 引数 ...

  コマンドを子プロセスとしてバックグラウンドで実行する。コマンドからの出力は無視する。

  (例) xterm を子プロセスとしてバックグラウンドで実行する

```
FORK xterm
OK
```

- HELP

  ヘルプメッセージとして XTS 命令の一覧を出力する。

```
HELP
CALL                CD                  DATA
ECHO                ENTRY               EVAL
EXEC                EXIT                HELP
.....
OK
```

- IF unix コマンド

  unix コマンドを実行し，終了コードが 0 ならば真，そうでなければ偽と判定する。

  (例) 数値の比較を行なう。

```
IF test 1 = 2
OK
THEN ECHO true
OK
ELSE ECHO false
false
```

- LOCK [{+|−}{echo|fatal|error|ok|all}] ...

  コールバック関数あるいはアクションから実行された XTS 命令のメッセージの出力を一時的にロックし XTS に入力された命令に対するメッセージ出力を優先するためのロックモードを指定する。ロックされたメッセージはロックが解除された時に出力される。+ はロックを掛けることを − はロックを解除することを指定し，echo, fatal, error, ok は結果コードがそれぞれに対応するメッセージのみにロック操作を行ない，all は全てのメッセージにロック操作を行なうことを指定する。初期は，LOCK −all であり，ロックは掛かっていない。なお，引数がない時は現在のロックモードの状態を出力する。

  (例) 結果コードが Error, OK のメッセージにロックを掛け，コールバック関数等で実行された命令に対するメッセージの出力を一時停止する。

```
LOCK +error +ok
OK
```

- MSG [{+|-}{echo|fatal|error|ok|all}] ...

  XTS 命令に対するメッセージのうち必要なメッセージのみを出力するためにメッセージ出力モードを指定する。+ はメッセージを出力することを - はメッセージを出力しないことを指定し，echo, fatal, error, ok は結果コードがそれぞれに対応するメッセージのみに出力操作を行ない，all は全てのメッセージに出力操作を行なうことを指定する。初期は MSG +all であり，全てのメッセージを出力する。なお，引数がない時は現在のメッセージ出力モードの状態を出力する。

  (例) 結果コードが OK の XTS 応答メッセージを出力しない。

```
MSG -ok
<メッセージは返らない>
```

  wait モードで実行する時は，EXEC 'MSG -ok' 'ECHO OK' とする必要がある。

- QUIT

  XTS を終了する。クライアントとの接続を切断する。

```
QUIT
```

- READ

  入力を待ち，入力されたデータを位置変数に代入する。

  (例) a b c を入力する。

```
READ
a b c                          <a b c を入力>
a b c                          $0: a, $1: b, $2: c
```

- SET 変数名 値

  名前付き変数に値を代入する。$変数名 で値を参照できる。

  (例) 変数 APPCLASS に XTS を代入する。

```
SET APPCLASS XTS
OK
ECHO $APPCLASS
XTS
```

- SETENV 環境変数名 [値]

  環境変数に値を設定する。環境変数の設定に失敗した時はエラーとなる。値を省略した時は現在の値を返す。

  (例) 環境変数 DISPLAY に unix:0 を代入する。

```
SETENV DISPLAY unix:0
OK
```

- SHIFT

  位置変数を左に一つシフトする。

  (例) 位置変数に a, b, c を代入し，シフトする。

```
DATA a b c
a b c
SHIFT
OK
ECHO $*
b c
```

- THEN XTS 命令

  直前の IF 命令が真となった時に，XTS 命令を実行する。

  (例) 真の場合に ECHO 命令を実行する。

```
THEN ECHO Hello World
Hello World                    真の場合
OK                             偽の場合
```

- UNPACK データ名 データ型 ...

  指定されたデータ名を各データ型からなる構造体へのポインタと解釈し，文字列に変換し出力する。データ型としては，整数，2バイト整数，実数，倍精度実数，文字列，文字をそれぞれ int, short, float, double, string, char で指定する。

  (例) コールバック関数の第三引数 client_data はデータ名 NULL に代入されるが，このデータが整数 10 へのポインタであった時には，以下のようになる。

```
UNPACK NULL int
OK 10
```

- VERSION

  XTS のメジャー番号，マイナー番号，その他バージョンに関する情報を表示する。

```
VERSION
OK 0 2 (93/06/06) by T.Takahama, Fukui Univ., JAPAN
```

Xt 命令

Xt 命令は，Xt 関数から Xt を除いた名前を持ち，以下の命令が使用可能である。

- AddCallback ウィジット名 コールバックラベル データ指定 XTS 命令 ...

  ウィジェット名に対応するウィジェットのコールバック関数として XTS 命令列 (XTS 命令 ...) を登録する。データ指定には XTS 命令列を実行する際の位置変数を指定する。データを必要としない時は NULL を指定する。NULL 以外の場合は XTS 命令として一度実行され，実行結果として得られた位置変数が XTS 命令列を実行する際の位置変数となる。したがって，文字列を渡す場合は DATA 命令，XTS 命令の実行結果を渡す場合は EVAL 命令を利用する。ただし，この時メッセージは出力しない。コールバックラベルで指定されたイベントが起きる度毎に，イベントが起こったウィジェット名を $0 に，データ指定を $1 以降に代入して，XTS 命令列が実行される。

  (例) YES ウィジェットの callback コールバック関数として，ウィジェット名などを出力する XTS 命令を登録する。

```
AddCallback YES callback "DATA say yes" 'ECHO $0 $1'
<ウィジェット YES で callback イベント発生>
YES say                          $0: YES, $1: say, $2: yes
```

  (注)XTS 命令で $ 変数を使用する際は，' で囲むか \ を付ける必要がある。

- AppAddAction アプリケーションコンテクスト名 アクション名 XTS 命令 ...

  XTS では標準的なアクションとして，XTS というアクションを提供している。この命令は XTS アクションから呼び出されるサブアクションとして XTS 命令列を登録する。アクションが呼ばれた際には，アクションが起こったウィジェット名を $0 に，アクションの引数を $1 以降に代入し，XTS 命令列が実行される。

  (例) Enter イベントに対する XTS アクションとして，"enter action" を宣言し，XTS サブアクション enter としてウィジェット名等を出力する XTS 命令列を登録する。

```
AppInitialize appcon APPCLASS NULL \
    "*translations: #override \\n <Enter>: XTS(enter action)"
OK mmm
CreateManagedWidget label labelWidgetClass
OK nnn
AppAddAction appcon enter 'ECHO $0 $1'
...
<ウィジェット label で <Enter> アクション発生>
label action
```

- AppAddTimeOut アプリケーションコンテクスト名 時間指定 データ指定
  XTS 命令 ...

  アプリケーションコンテクストに時間経過により実行される XTS 命令を登録する。時間指定はミリ秒単位で指定するが，+ミリ秒と指定した時はその時間毎に繰り返し起動される。データ指定，XTS 命令については AddCallback 命令と同様である。

  (例) デフォルトのアプリケーションコンテクストに 1 秒毎に date コマンドを繰り返す XTS 命令を登録する。

```
AppAddTimeOut NULL +1000 NULL 'CALL date'
OK
```

- AppCreateShell ウィジェット名 アプリケーション名 シェルクラス名
  親ウィジェット名 [リソース名 リソース値] ...

  親ウィジェットの子供としてシェルクラス名で指定したシェルウィジェットを生成し，各リソースに値を設定する。

  (例) ウィジェット dialog の子供に topLevelShellWidgetClass のウィジェット popup をアプリケーション名 Popup で作成する。

```
AppCreateShell popup Popup topLevelShellWidgetClass dialog
OK mmm
```

  (注) ウィジェットを生成した後にリソースに値を設定するので，生成時の設定とは動作が異なる場合がある。

- AppInitialize アプリケーションコンテクスト名 アプリケーションクラス名
  コマンドライン引数 [NULL フォールバックリソース指定 ...
  [NULL [リソース名 リソース値] ...]]

  アプリケーションコンテクストとアプリケーションシェルウィジェットを作成する。アプリケーションクラスがルートウィジェット名となる。コマンドライン引数，フォールバックリソース指定，リソース指定をこの順に NULL で区切って記述することができる。

  (例) ホスト remote に Hello クラスのアプリケーションコンテクスト appcon およびウィジェット Hello を作成する。フォールバックリソースおよびリソースも指定する。

```
AppInitialize appcon Hello -display remote:0 NULL \
        "*Label.label: Hello" "*Dialog.label: Welcome" NULL \
        x 100 y 100
OK mmm
```

- AppMainLoop アプリケーションコンテクスト名

  指定したアプリケーションコンテクストでイベント待ちループに入る。

  (例) アプリケーションコンテクスト appcon でイベント待ちループに入る。

```
AppMainLoop appcon
OK
```

(注) これ以後の XTS の処理は Input Event として登録され実行を継続する。

- AppSetFallbackResources アプリケーションコンテクスト名
  フォールバックリソース指定 ...

  アプリケーションコンテクストにフォールバックリソースを指定する。

  (例) アプリケーションコンテクスト **appcon** にクラス **Label** のリソース **label** を指定するフォールバックリソースを登録する。

```
AppSetFallbackResources appcon "*Label.label: Hello"
OK mmm
```

- CallCallbacks ウィジェット名 [コールバックラベル]

  ウィジェットにおいてコールバックラベルで登録されたコールバック関数を呼び出す。コールバックラベルが省略された時は **callback** が指定されたものと見なす。

  (例) **yes** ウィジェットの **callback** で登録されたコールバック関数を呼び出す。

```
CallCallbacks yes callback
OK
```

- CloseDisplay ディスプレイ名

  OpenDisplay 命令でオープンしたディスプレイを閉じる。

  (例) ディスプレイ **display** を閉じる。

```
CloseDisplay display
OK
```

- CreateApplicationContext アプリケーションコンテクスト名

  アプリケーションコンテクストを作成する。

  (例) アプリケーションコンテクスト **appcon** を作成する。

```
CreateApplicationContext appcon
OK
```

- CreatePopupShell ウィジェット名 シェルクラス名 親ウィジェット名

  親ウィジェットの子供としてシェルクラスで指定したポップアップシェルを作成する。

  (例) ウィジェット **dialog** の子供として **topLevelShellWidgetClass** のポップアップシェルウィジェット **popup** を作成する。

```
CreatePopupShell popup topLevelShellWidgetClass dialog
OK
```

- Create(Managed)Widget ウィジェット名 ウィジェットクラス 親ウィジェット名
  [リソース名 リソース値] ...

  ウィジェットを作成する。Managed の場合は，マネージする。

  (例) ウィジェット box の子供として labelWidgetClass のウィジェット label を作成する。

```
CreateWidget label labelWidgetClass box
OK mmm
```

- DestroyWidget ウィジェット名 ...

  ウィジェットを破壊する。

  (例) ウィジェット label を破壊する。

```
DestroyWidget label
OK
```

- GetValues ウィジェット名 リソース名 ...

  指定したリソース名に対するリソース値をウィジェットから取り出す。存在しないリソース名を指定したり，文字列に変換できない型のリソース名を指定した場合には，NULL が返される。

  (例) ウィジェット label のリソース x, y, undefined の値を求める。

```
GetValues label x y undefined
OK 100 100 NULL
```

- Initialize ウィジェット名 アプリケーションクラス名 コマンドライン引数
  [NULL [リソース名 リソース値] ...]

  AppInitialize と同様にルートウィジェットを作成する。コマンドライン引数，リソース指定をこの順に NULL で区切って記述することができる。なお，同時にデフォルトのアプリケーションコンテクスト名 NULL が登録される。

  (例) アプリケーションクラスが Hello のルートウィジェット hello を作成する。

```
Initialize hello Hello NULL x 100 y 100
OK mmm
```

- MainLoop

  デフォルトのアプリケーションコンテクストでイベント待ちループに入る。

```
MainLoop
OK
```

- ManageChild(ren) ウィジェット名 ...

  ウィジェットをマネージする。

  (例) ウィジェット label をマネージする。

```
ManageChild label
OK
```

- MapWidget ウィジェット名 ...

  ウィジェットをマップする。

  (例) ウィジェット label をマップする。

```
MapWidget label
OK
```

- OpenDisplay ディスプレイ名 アプリケーションコンテクスト名 ディスプレイ指定 アプリケーション名 アプリケーションクラス名 コマンドライン引数

  ディスプレイ指定に基づきアプリケーションコンテクスト上にアプリケーション名・クラス名で指定されたディスプレイをオープンする。コマンドライン引数も指定できる。

  (例) ホスト remote に対し，アプリケーションコンテクスト appcon 上に，Hello クラスのアプリケーション hello 用ディスプレイ remote_disp を開く。

```
OpenDisplay remote_disp appcon remote:0 hello Hello
OK mmm
```

- Parent ウィジェット名

  ウィジェットの親ウィジェットを返す。親のウィジェット名が登録されていない場合には無名ウィジェット名 (_ウィジェット ID) を返す。

  (例) ウィジェット dialog の子供が label の場合に label の親を返す。

```
Parent label
OK dialog
```

- Popdown ウィジェット名

  ウィジェットをポップダウンする。

  (例) ウィジェット poped_up をポップダウンする。

```
Popdown poped_up
OK
```

- Popup ウィジェット名 グラブ種類名

  ウィジェットを指定されたグラブ種類でポップアップする。(Xt)GrabNone, (Xt)GrabNonexclusive, (Xt)GrabExclusive というグラブ種類が指定できる。

  (例) ウィジェット poped_up を GrabNone でポップアップする。

```
Popup poped_up GrabNone
OK
```

- RealizeWidget ウィジェット名

  ウィジェットを実現する。

  (例) ウィジェット toplevel を実現する。

```
RealizeWidget toplevel
OK
```

- SetValues ウィジェット名 | リソース名 リソース値 | ...

  指定した各リソース名に対するリソース値をウィジェットに設定する。

  (例) ウィジェット label のリソース x, y に 100 を設定する。

```
SetValues label x 100 y 100
OK
```

- ToolkitInitialize

  ツールキットを初期化する。

```
ToolkitInitialize
OK
```

- UnmanageChild(ren) ウィジェット名 ...

  ウィジェットをマネージから外す。

  (例) ウィジェット label をマネージから外す。

```
UnmanageChild label
OK
```

- UnmapWidget ウィジェット名 ...

  ウィジェットをアンマップする。

  (例) ウィジェット label をアンマップする。

```
Unmap label
OK
```

Xaw 命令

- XawDialogAddButton 親ウィジェット名 ボタンラベル データ指定 XTS 命令 ...

  ダイアログクラスの親ウィジェットの子供としてボタンラベルで指定したラベルを持つコマンドクラスのウィジェットを作成し，callback として XTS 命令列を登録する。すなわち，

  CreateManagedWidget 子ウィジェット名 commandWidgetClass
  親ウィジェット名
  AddCallback 子ウィジェット名 callback データ指定 XTS 命令 ...

  と同じ処理をする。ただし，作成された子ウィジェットにはウィジェット名が付かない。

  (例) ウィジェット dialog に yes ボタンを追加し，yes を出力するコールバック関数を登録する。

```
XawDialogAddButton dialog yes NULL "ECHO yes"
OK
```

- XawDialogGetValueString ウィジェット名

  ダイアログクラスのウィジェットに入力されたテキストを返す。

  (例) ダイアログウィジェット dialog に Hello が入力されていた場合に入力されたテキストを返す。

```
XawDialogGetValueString dialog
OK Hello
```

- XawListChange ウィジェット名 リスト値 numberStrings 値 longest 値 再表示フラグ

  リストクラスのウィジェットの list, numberStrings, longest の値を指定するとともに，再表示を行なうかどうかを 0, 1 で指定する。

  (例) list ウィジェットの list 値を "a", "b", "c" とし，それに基づき再表示する。

```
XawListChange list "a b c" 0 0 1
OK
```

- XawListShowCurrent ウィジェット名

  リストクラスのウィジェットで現在選択されている項目を返す。

  (例) list ウィジェットで現在選択されている項目 b を得る。

```
XawListShowCurrent list
OK b
```

X 命令

X 命令ではグラフィクスコンテクスト (GC) を使用するため，GC 名が存在する。GC 名として NULL を指定すればデフォルトの GC を指定したことになる。

- XClearWindow ウィジェット名

  ウィジェットのウィンドウをクリアする。

  (例) ウィジェット label のウィンドウをクリアする。

```
XClearWindow label
OK
```

- XDrawLine ウィジェット名 GC 名 x0 y0 x1 y1

  指定したグラフィックスコンテクストでウィジェットの (x0,y0)−(x1,y1) に直線を引く。

  (例) ウィジェット label の対角線にデフォルトの GC で直線を引く。

```
EVAL GetValues label width height
OK 100 200                          $1: 100, $2: 200
XDrawLine label NULL 0 0 $1 $2
OK
```

- XDrawString ウィジェット名 GC 名 x y 文字列

  指定したグラフィックスコンテクストでウィジェットの (x,y) に文字列を描画する。

  (例) ウィジェット label の (10, 10) に Hello World を描画する。

```
XDrawString label NULL 10 10 "Hello World"
OK
```

- XFillRectangle ウィジェット名 GC 名 x0 y0 x1 y1

  指定したグラフィックスコンテクストでウィジェットの (x0,y0)−(x1,y1) を対角線とする長方形の内部を塗りつぶす。

  (例) ウィジェット label を塗りつぶす。

```
EVAL GetValues label width height
OK 100 200                          $1: 100, $2: 200
XFillRectangle label NULL $1 $2
OK
```

- XFlush ウィジェット名

  指定したウィジェットが属するディスプレイの出力バッファの内容を強制的に処理する。

```
XFlush label
OK
```

- XSync ウィジェット名 廃棄

  指定したウィジェットが属するディスプレイの出力バッファの内容を強制的に処理する。廃棄が偽ならば内容を廃棄しない。

```
XSync label 1
OK
```

List of Major Publications

[1] 高濱徹行, 中谷洋一, 小倉久和, 中村正郎: 応用プログラム群を統合するウィンドウ型ユーザインタフェースシステム: UAI/X, 電子情報通信学会論文誌, Vol. J75-D-I, No. 7, pp. 479-487 (1992)

[2] 小倉久和, 魚見勇治, 高濱徹行, 小高知宏: 対話的初級英語学習支援システムにおける文章知識の表現, CAI学会誌, Vol. 11, No. 1, pp. 3-13 (1994)

[3] 高濱徹行, 小倉久和, 中村正郎: テキスト型アプリケーション群を統合するグラフィカルユーザインタフェースシステム: XTSS, 電子情報通信学会論文誌, Vol. J77-D-I, No. 7, pp. 493-502 (1994)

[4] T.Odaka, T.Takahama, H.Wagatsuma, K.Shimada, H.Ogura: A visual data analysis system for the medical image processing, Journal of MEDICAL SYSTEMS, Vol. 18, No. 3, pp. 167-173 (1994)

[5] 高濱徹行, 阪井節子, 小倉久和, 中村正郎: 強化学習法による離散値制御のためのファジィ制御規則の学習, 日本ファジィ学会誌, Vol. 8, No. 1, pp. 115-122 (1996)

[6] T.Takahama, S.Sakai: LEARNING FUZZY RULES FOR BANG-BANG CONTROL BY REINFORCEMENT LEARNING METHOD, Proceedings of 1996 International Fuzzy Systems and Intelligent Controls Conference, pp.193-202 (1996)

[7] ムハマドロムジ, 高濱徹行, 小高知宏, 小倉久和: 倒立二重振子系に対するファジィ制御知識の表現とスケーリングによる適応制御, 日本ファジィ学会誌, Vol. 8, No. 3, pp.576-585 (1996)

[8] 高濱徹行, 牧野とみ, 阪井節子: CAIシステムにおけるグラフィカル・ユーザインタフェースの分離手法, 教育システム情報学会誌, Vol.13, No.4 (1997) 掲載予定

List of Other Publications

[1] 高濱徹行, 田中伸佳, 中村順一, 長尾真, 堤泰治郎, 藤崎哲之助: prologによる自然言語解析, IBM 東京サイエンティフィック・センター・レポート (1982)

[2] 高濱徹行, 長尾真: データベース・アクセスのための自然言語インタフェースにおいて解析ルールを獲得する枠組み, 情報処理学会第32回全国大会講演論文集, pp. 1275-1276 (1986)

[3] 高濱徹行: パーソナル・コンピュータ上の大型計算機用ユーザ・インタフェース・システムの開発, 情報処理学会第36回全国大会講演論文集, pp. 1379-1380 (1988)

[4] 高濱徹行: OPTION: コマンドライン解析ルーチン自動生成プログラム, 情報処理学会第38回全国大会講演論文集, pp. 777-778 (1989)

[5] 高濱徹行, 中谷洋一, 小倉久和, 中村正郎: 応用プログラム群を統合するウィンドウ型ユーザインタフェースシステム, 平成2年度電気関係学会北陸支部連合大会, p.119 (1990)

[6] 高濱徹行, 中谷洋一, 小倉久和, 中村正郎: 応用プログラム群を統合するウィンドウ型ユーザインタフェースシステム: UAI/X, 情報処理学会第42回全国大会 (5), pp.33-34 (1991)

[7] 高濱徹行: 倒立振子制御ファジィ制御規則の学習について, 北信越ファジ研究会ミニシンポジウム, p. 26 (1991)

[8] 高濱徹行, 宮本誠司, 小倉久和, 中村正郎: 遺伝アルゴリズムを応用したファジィ制御規則の学習について, 平成3年度電気関係学会北陸支部連合大会講演論文集, p. 113 (1991)

[9] 高濱徹行, 宮本誠司, 小倉久和, 中村正郎: ファジィ推論による制御学習—遺伝アルゴリズムによるルール学習—, 情報処理学会第43回全国大会講演論文集 (3), pp. 91-92 (1991)

[10] 高濱徹行, 宮本誠司, 小倉久和, 中村正郎: 遺伝アルゴリズムによるファジィ制御規則の学習, 日本ファジィ学会中部支部・北信越ファジィ研究会合同研究会講演論文集, pp. 15-19 (1992)

[11] 高濱徹行, 小倉久和, 中村正郎: 自然言語インタフェースにおける文法解析規則の詳細化について, 情報処理学会第44回全国大会講演論文集 (3), pp. 233-234 (1992)

[12] 高濱徹行, 宮本誠司, 小倉久和, 中村正郎: 遺伝アルゴリズムによるファジィ制御規則の学習, 第8回ファジィシンポジウム講演論文集, pp. 241-244 (1992)

[13] 高濱徹行: 遺伝的アルゴリズムとその応用, 日本ファジィ学会北信越支部研究会, pp. 1-7 (1992)

[14] 加藤直樹, 高濱徹行, 小倉久和: あいまいな量的属性のファジィ言語表現の特徴とその検討, 福井大学工学部研究報告, Vol. 41, No. 1, pp. 73-85 (1993)

[15] 高濱徹行, 八木正和, 小倉久和, 中村正郎: グラフィカルユーザインタフェース構築支援ツールの開発, 福井大学工学部研究報告, Vol. 41, No. 2, pp. 199-208 (1993)

[16] 臼井秀宣, 高濱徹行, 小高知宏, 館清隆, 小倉久和: 英文コーパス検索における語の文法機能の利用, 福井大学工学部研究報告, Vol. 41, No. 2, pp. 209-222 (1993)

[17] 高濱徹行, 八木正和, 小倉久和, 中村正郎: ウィンドウ型ユーザインタフェース記述システム: XTSS, 平成5年度電気関係学会北陸支部連合大会, p.269 (1993)

[18] 高濱徹行: Xツールキットサービスプロトコルによる Xウィンドウプログラミング, 福井大学情報処理センター NETWORK, Vol.7, No.1, pp.19-44 (1993)

[19] 高濱徹行, 小倉久和, 中村正郎: ニューラルネットワークによる制御のための規則獲得について, 情報処理学会第46回全国大会講演論文集 (2), pp. 267 268 (1993)

[20] 高濱徹行, 八木正和, 小倉久和, 中村正郎: ウィンドウ型ユーザインタフェース記述システム: XTSS, 平成5年度電気関係学会北陸支部連合大会講演論文集, p. 269 (1993)

[21] 笹田国博, 小高知宏, 高濱徹行, 小倉久和: 算数問題文の解析とその知識表現, 福井大学工学部研究報告, Vol. 42, No. 1, pp. 53 66 (1994)

[22] 高濱徹行, 水船博康, 小倉久和, 中村正郎: 分解結合型遺伝的アルゴリズムによる巡回セールスマン問題の解法, 情報処理学会第48回全国大会講演論文集 (2), pp. 247 248 (1994)

[23] 高濱徹行: 試行錯誤によるファジィ制御規則の獲得について, 日本ファジィ学会北信越支部第3回ファジィミニシンポジウム, pp. 63-66 (1994)

[24] 阪井節子, 佐藤和枝, 佐藤直子, 高濱徹行, 宮阪憲治: 低学年算数問題文章における自然言語理解について, 福井大学教育学部紀要 第V部 応用科学 (技術編), Vol. 28, pp. 1-19 (1995)

[25] 高濱徹行: 正負のスキーマに基づく遺伝的アルゴリズム, 情報処理学会第50回全国大会講演論文集 (2), pp. 277-278 (1995)

[26] 高濱徹行, 阪井節子, 飯塚由紀子, 佐藤直子: 初等算数文章題のためのCAIシステム, 言語処理学会第1回年次大会発表論文集, pp. 225-228 (1995)